Mensurasoft :

Système de pilotes d'appareils de mesure de type "Bibliothèques dynamiques", pour Windows et Linux

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

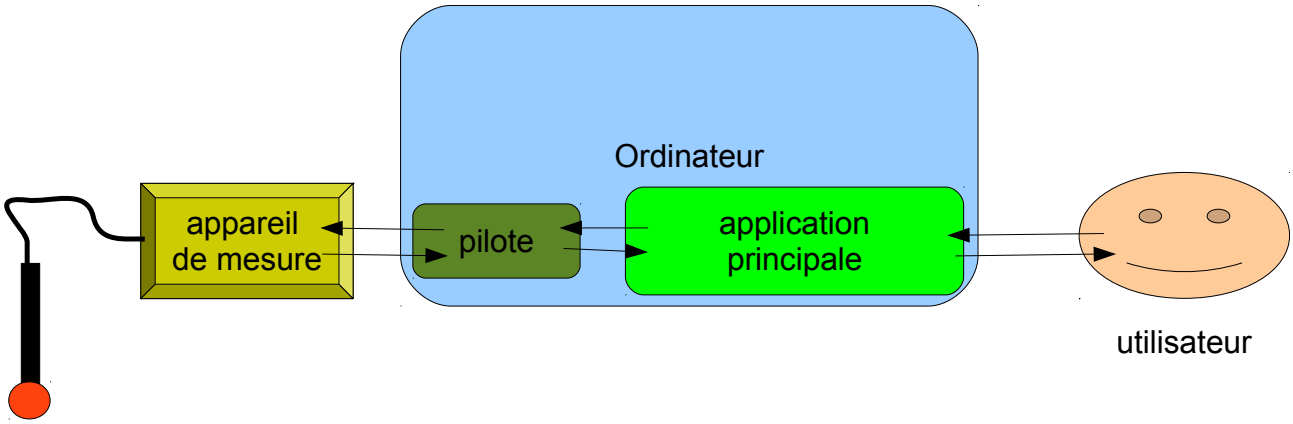

Table des matières

Pourquoi le "système Mensurasoft" ?

Il y a une grande diversité dans le matériel de mesure connectable à un ordinateur : certains appareils sont des cartes à enfoncer à l'intérieur de l'ordinateur, d'autres sont des appareils à brancher sur une prise sérielle RS232, d'autres sur la prise parallèle, ou bien sur la prise "manettes de jeux", ou sur une prise USB, etc. Certains appareils ne remplissent qu'une fonction, par exemple des pHmètres, d'autres peuvent faire deux types de mesures, par exemple à la fois le pH et la température. Certains peuvent faire des mesures de divers types, mais une seule à la fois, comme les multimètres. Certains systèmes multifonctions (Jeulin ESAO, Orphy, Arduino,...) peuvent réaliser simultanément des mesures sur divers capteurs, et peuvent agir sur le système de mesure en actionnant des relais ou en modifiant la tension (en volts) d'un connecteur spécifique.

Lorsqu'un programmeur rédige un logiciel de mesure, il ne peut pas le concevoir d'emblée capable de fonctionner avec tous les appareils ou interfaces de mesure. Même si il a la liste de tous les appareils existant actuellement, il ne peut pas prévoir ceux qui seront inventés l'année prochaine. Il faut donc qu'il rédige un logiciel "adaptable", qui pourra s'adapter aux appareils à venir.
C'est le même problème que pour les logiciels de traitement de texte et les imprimantes : il est impossible de faire un logiciel qui tienne compte des particularités de toutes les imprimantes. C'est pourquoi ces logiciels fonctionnent avec des "pilotes d'imprimantes", ou "drivers", qui sont spécifiques du matériel en question.
La solution est donc semblable. On peut réaliser d'une part des "pilotes d'appareils de mesure", spécifiques de chaque appareil, ou même correspondant à un type d'emploi particulier d'un appareil de mesure. D'autre part, on peut faire des logiciels d'application, qui tracent des graphiques des mesures réalisées, qui font des calculs statistiques complexes sur ces mesures, les écrivent et les lisent dans des fichiers. Les logiciels d'application utilisent les fonctions de mesure des pilotes d'appareils de mesure.
Le système présenté ici est volontairement simple. Il ne permet pas toujours d'utiliser toutes les capacités des appareils, mais il est utilisable avec un très grand nombre d'appareils, sous les deux principaux systèmes d'exploitation GNU-Linux et Microsoft-Windows (et probablement Mac-OS), et la programmation peut être faite avec des langages de programmation très divers.
Les pilotes proposés ici seront des bibliothèques dynamiques, c'est à dire des fichiers contenant du code exécutable, mais qui ne peuvent pas s'exécuter de façon autonome. Les bibliothèques dynamiques doivent être appelées par des programmes exécutables (ici appelés "logiciels d'application", ou "application"), qui peuvent utiliser leurs fonctions spécifiques.

Le logiciel d'application lui-même appellera les fonctions du pilote (lecture des entrées et commande des sorties...). Il ne se posera pas le problème de savoir comment faire les mesures : il enverra simplement l'ordre au pilote de faire la mesure et de lui renvoyer le résultat. Ensuite, c'est ce logiciel d'application qui utilisera les résultats pour les présenter à l'utilisateur, soit sous forme d'un tableau, soit sous forme d'un graphique, soit sous forme d'un cadran à aiguille, soit par une alarme qui se déclenche lorsqu'un seuil est dépassé, etc.

Ce système fonctionne correctement depuis plusieurs années avec les logiciels de la famille Mesugraf pour Windows.
Le premier logiciel a été développé en Delphi 1, qui est un langage de type Pascal, conçu pour les versions 16 bits de Windows (Windows_3.1). Il fonctionnait avec un grand nombre d'appareils de mesures (Orphy GTS et portable, Jeulin ESAO3, ADES, Pierron SMF10 et Expert, et un grand

nombre de multimètres, luxmètres, thermomètres, balances... à brancher sur une prise sérielle). Les DLL développées pour Windows 16 bits ne sont pas utilisables avec les systèmes Windows 32 bits (95, 98, XP...). Bien que basées sur le même principe, les librairies pour ces systèmes doivent être légèrement modifiées et recompilées pour pouvoir fonctionner.
Le logiciel MGW32 a été développé en suivant les mêmes bases en Delphi 5. Il utilise des bibliothèques dynamiques 32 bits.
En 2011, Mensurasoft-PB a été programmé en PureBasic, pour Windows et pour Linux.
En 2012, Mensurasoft-LZ a été programmé en Lazarus/FreePascal, pour Windows et pour Linux

Le système de pilotes « Mensurasoft » est conçu pour Windows et Linux. Ils peuvent être réalisés en Basic, C/C++ ou Pascal, et utilisés par de nombreux langages.

1 Fonctions souhaitables des pilotes, et quelques contraintes

De nombreux langages de programmation sont "sensibles à la casse", c'est à dire qu'ils font la distinction entre majuscules et minuscules dans le nom des identificateurs, et en particulier dans les noms de fonctions. Par exemple, une variable nommée "mavariable" sera différente d'une variable nommée "MaVariable". Au contraire, d'autres langages ne font pas une telle différence.
La règle employée ici est de **nommer les fonctions en lettres minuscules**.
Les valeurs numériques entières seront de type "signé", codé sur 4 octets ("longint" des langages Pascal, "int" des langages C/C++,
Les valeurs réelles seront codées sur 8 octets ("double précision").
Les chaînes de caractères sont de type "pointeur de chaîne à zéro terminal".
Fondamentalement, ce système peut fonctionner aussi bien avec le codage ANSI (1octet par caractère) qu'avec le codage Unicode (2 octets par caractère) mais dans la pratique, pour éviter les incompatibilités entre le programme d'application et le pilote, le mieux est d'utiliser le codage ANSI.

1.1 Fonctions numériques d'entrée et de sortie

Un appareil de mesure doit envoyer des mesures à l'ordinateur. par des fonctions d'entrées (les nombres entrent dans l'ordinateur à partir de l'appareil). Elles sont notées par la lettre E (comme "Entrée", "Entrance", "Eingang", "Entrada")
Souvent, l'ordinateur peut commander à l'appareil de changer d'état, par exemple allumer une lampe ou un moteur.. Il faut donc des fonctions de sortie (les nombres sortent de l'ordinateur pour aller dans l'appareil, et modifier son fonctionnement). Elles sont notées par la lettre S (comme "Sortie", "Salida").
Ces fonctions d'entrée et de sortie peuvent être binaires ou analogiques.
"Binaires" signifie qu'elles ne peuvent prendre seulement deux états (0/1), et on note ces fonctions par la lettre B.
"Analogiques" signifie qu'elles peuvent prendre une multitude de valeurs, par exemple une tension en volts ou millivolts, une température en degrés Celsius, une valeur de pH. On les note par la lettre A. Souvent, ces valeurs analogiques sont fondamentalement des nombres entiers positifs, donnés par un convertisseur analogique/numérique. Dans d'autres cas, les valeurs analogiques peuvent être des "nombres réels", positifs ou négatifs, et qui ne sont pas des valeurs entières ;, en général, en informatique, ces nombres réels sont codés en "double précision", et on note ce dernier cas par la lettre D (comme "Double").
On a donc des fonctions notées eb, ea, ead, sb, sa, sad, qui portent un numéro. En suivant une convention habituelle en informatique, les numéros commencent à zéro. eb(0) sera donc la première entrée binaire, et sad(1) la seconde sortie analogique, à laquelle on fixera une valeur codée comme un nombre réel
Normalement, ces numéros sont des nombres entiers, ainsi que les paramètres des fonctions binaires, alors que les entrées et sorties analogiques peuvent avoir des paramètres de type réel codés en double précision. Certains logiciels n'acceptent de travailler qu'avec des nombres de type "double précision" ; pour ce cas, on peut faire des fonctions se terminant par "double" : pour ces logiciels, on a ainsi les fonctions eadouble, ebdouble, sadouble, sbdouble (peu important).

1.2 Noms de ces fonctions

Lorsque l'appareil n'a qu'une fonction, on ne peut utiliser que celle-ci ; par exemple avec un pHmètre, on ne peut que mesurer le pH. Lorsqu'un appareil a plusieurs fonctions, on doit choisir

une de ces fonctions parmi plusieurs.
Il est pratique d'associer une chaîne de caractère à chaque fonction numérique : ce sera le nom de la fonction. Lorsque le nom existe, cela signifie que la fonction numérique existe ; lorsque le nom n'existe pas (chaîne de caractères de longueur nulle), cela signifie que la fonction numérique n'existe pas. Ici, les fonctions donnant le nom des fonctions numériques sont notées par la lettre n : nead(0) signifie le nom de la première voie d'entrée analogique donnant un nombre réel, et nsb(1) est le nom de la deuxième sortie binaire.
Ainsi, on pourra faire apparaître ces noms dans des menus déroulants et des boites de dialogue.

1.3 Titre de l'appareil, et détail de cet appareil

La fonction "titre" sera une petite chaîne de caractères donnant le titre de l'appareil de mesure, que l'on pourra mettre au début de la boit de dialogue destinée au choix des voies.
"detail" sera une chaîne de caractères plus longue, pouvant donner des informations supplémentaires par rapport à "titre" : nom du programmeur, date de réalisation du pilote, etc.

1.4 Calibration : une fonction facultative

Certains appareils peuvent nécessiter des réglages supplémentaires. Par exemple, un appareil mesurant la teneur du milieu en dioxygène nécessitent de "faire le zéro" en étalonnant la sonde, ou bien un spectrophotomètre nécessite de "faire le blanc" sur un témoin non coloré avant de mesurer une coloration. Dans ces cas là, il est prévu une fonction "calibration", pour ouvrir des boites de dialogue permettant ces réglages. Cette fonction reçoit une chaîne en paramètre, et renvoie une chaîne, mais normalement ces paramètres n'ont aucune importance.
De nombreux appareils peuvent être réglés sans avoir besoin d'une telle fonction, par exemple en utilisant des fonctions fictives : une sortie binaire peut correspondre au choix d'un calibre, ou bien deux sorties analogiques peuvent permettre de régler le zéro et la pente d'un appareil.

1.5 Deux types de passage de paramètres : stdcall et cdecl

Pour que le programme principal et le pilote puissent échanger leurs informations, il faut qu'ils utilisent la même convention pour le passage des paramètres des fonctions. Malheureusement, il existe différentes possibilités, et les deux principales sont "stdcall" et "cdecl".
Pour que ce système de pilotes puisse fonctionner avec tous les langages de programmation, il faut qu'il puisse utiliser les deux types. De nombreux langages de programmation peuvent utiliser les deux conventions, mais certains ne peuvent en utiliser qu'un seul type. Dans le monde Windows, divers langages (Panoramic, FnxBasic...) ne peuvent utiliser que les fonctions stdcall, et les compilateurs C et C++ font beaucoup plus simplement les fonctions cdecl que les fonctions stdcall.
Les fonctions utilisant la convention cdecl commenceront par la lettre c, et celles utilisant la convention stdcall commenceront par std. Elles font exactement la même chose, et ne se distinguent que par le type de passage des paramètres.
On aura donc stdead et cead, stdeb et ceb, etc.

1.6 Attention à l'incompatibilité entre les bibliothèques dynamiques "32 bits" et "64 bits"

Les micro-ordinateurs ("PC") ont un microprocesseur travaillant sur un certain nombre de bits simultanément. Les premiers Apple II travaillaient sur 8 bits, les PC de 1990 travaillaient sur 16 bits, ceux de 2000 fonctionnaient sur 32 bits, et les nouveaux ordinateurs peuvent fonctionner avec 64 bits.
Pour exploiter au mieux ces microprocesseurs, les systèmes d'exploitation évoluent aussi, ainsi que

les logiciels pouvant fonctionner avec ces systèmes d'exploitation.
Depuis Windows95 jusqu'à WindowsXP, les systèmes de Microsoft étaient en 32 bits, mais Windows7 est en 64 bits. Dans le monde Linux, actuellement (2012), on diffuse simultanément des versions 32 bits et 64 bits. Normalement, les systèmes d'exploitation en 64 bits peuvent quand même faire fonctionner les programmes compilés pour 32 bits.
De même, les compilateurs peuvent fonctionner soit en 32 bits, soit en 64 bits. Par exemple, PureBasic existe dans deux versions "x86" et "x64", pour chaque type de système d'exploitation. Une bibliothèque dynamique 32 bits ne sera pas appelable par un programme principal conçu pour 64 bits, et une bibliothèque dynamique en 64 bits ne sera pas appelable par un programme en 32 bits.
Pour l'expérimentation par ordinateur, jusqu'à maintenant, il n'y a pas d'avantage net à faire des programmes pour 64 bits. Les logiciels développés jusqu'à maintenant sont conçus en 32 bits ("x86"). C'est donc la même chose pour les pilotes d'appareils de mesure, qui doivent être compatibles avec ces logiciels : ils doivent être compilés pour 32 bits.
Néanmoins, le principe de ces bibliothèques dynamiques est valable aussi bien en 32 bits qu'en 64 bits : il faut simplement les compiler avec des programmes différents.

Syntaxe des fonctions des pilotes

«e» = entrée
(binaire ou analogique)
Il peut exister aussi des fonctions de sortie «s», binaires ou analogiques, qui déclenchent des actions.

«a» = analogique : les valeurs renvoyées peuvent prendre différentes valeurs, par exemple une température, ou une longueur.
Il peut exister aussi des mesures ou des actions binaires «b», codées par 0 ou 1.

«d» indique que le résultat renvoyé sera un réel de type double précision. Pour les sorties analogiques, le 2e paramètre, qui fixe la valeur de la sortie, est aussi de type double (alors que le premier, qui indique le numéro de la voie, est de type entier). Lorsqu'il n'y a pas de lettre, le résultat renvoyé est de type entier codé sur 4 octets, et les paramètres d'entrée sont aussi des entiers codés sur 4 octets. Lorsque d est remplacé par «double», non seulement le résultat renvoyé est de type double, mais aussi le ou les paramètres sont de type double.

résultat renvoyé par la fonction : pour les fonctions d'entrée, c'est la mesure ; pour les fonctions de sortie, c'est la valeur choisie ; pour les fonctions de nom, c'est le nom

x=stdead(0)

Il existe plusieurs conventions d'appel pour la communication entre une bibliothèque dynamique et le programme principal.
«std» signifie stdcall, qui est la convention la plus fréquente sous Windows.
On peut aussi utiliser la convention cdecl, notée par «c», qui est la plus utilisée sous Linux.
Les autres conventions, «pascal» et «safecall» sont moins utilisées.

Le premier paramètre est le numéro de la voie (entrée ou sortie, binaire ou analogique). Sauf lorsqu'il y a le suffixe «double», c'est une valeur entière codée sur 4octets. La numérotation commence à zéro (la première voie est la voie 0).
Pour les sorties, il y a un deuxième paramètre, la valeur à fixer. Pour les sorties logiques, c'est 0 ou 1, pour les sorties analogiques, c'est une valeur variable, codée par un entier (pas de suffixe d ou double) ou par un réel de type double (suffixe d ou double)

x=stdnead(0)

Chaque fonction numérique a un nom, correspondant à une fonction ayant la lettre «n». Le paramètre d'entrée est du même type que celui de la fonction numérique, et le paramètre de sortie (valeur renvoyée) est une chaîne à zéro terminal. Lorsque cette chaîne est non nulle (chaîne de longueur supérieure à 0), alors la fonction numérique existe ; lorsque la chaîne renvoyée est de longueur nulle, c'est que la fonction numérique n'existe pas.

Autres fonctions, renvoyant des chaînes de caractères (à zéro terminal) :
«titre» (stdtitre, ctitre), sans paramètre : donne le titre de l'appareil (en bref)
«detail»(stddetail, cdetail), sans paramètre : donne le détail de l'appareil (plus long)

«calibration» (stdcalibration, ccalibration), avec un paramètre chaîne de caractère à zéro terminal : fonction facultative, qui permet la calibration de l'appareil, par erxemple régler le blanc d'un spectrophotomètre. Beaucoup d'appareils n'en ont pas besoin.

(Pierre Dieumegard, avril 2011)

2 Liste des fonctions

Ces fonctions existent toujours sous deux formes, cdecl (préfixe c) et stdcall (préfixe std). Il n'est indiqué ici que la forme stdcall.
La description est donnée ici avec la syntaxe du langage Pascal.

2.1.1 Fonctions fondamentales

Tous les appareils ou dispositifs expérimentaux n'ont pas obligatoirement les fonctions numériques décrites ci-dessous. Par exemple, un pHmètre n'a pas de sorties logiques ou analogiques, et pour ces fonctions de sortie, les noms des fonctions ont une longueur nulle, ce qui signifie que les fonctions numériques n'existent pas.

Inversement, on peut imaginer un pilote pour la commande d'un moteur, où il n'existe pas de fonctions d'entrée analogique ou binaire : il faut mettre les noms de ces fonctions d'entrée à une longueur nulle.

```
function stdea(n:longint) : longint ;stdcall;export;
```
nième entrée analogique (résultat directement renvoyé par le convertisseur analogique-numérique)

```
function stdnea(n:longint):pchar ;stdcall;export;
```
nom de la nième entrée analogique

```
function stdead(n:longint):double ;stdcall;export;
```
nième entrée analogique (résultat converti en unité SI, le plus souvent en Volts)

```
function stdnead(n:longint):pchar ;stdcall;export;
```
nom de la nième entrée analogique renvoyant le résultat sous forme de "double"

```
function stdsa(n:longint;valeur:longint):longint;stdcall;export;
```
envoi de valeur au convertisseur numérique analogique sur la nième sortie analogique ; si tout s'est bien passé, elle renvoie valeur.

```
function stdnsa(n:longint):pchar;stdcall;export;
```
nom de la nième sortie analogique fixant directement la valeur du CNA.

```
function stdsad(n:longint; valeur:double):double;stdcall;export;
```
Elle permet la fixation de la nième sortie analogique à valeur. Pour la plupart des interfaces, la valeur sera en volts, mais on peut imaginer des systèmes plus complexes, où une sortie puisse commander une température, une vitesse de rotation, une intensité lumineuse, ou d'autres grandeurs pouvant varier.

```
function stdnsad(n:longint):pchar;stdcall;export;
```
C'est le nom de la fonction précédente

```
function stdeb(n:longint):longint ;stdcall;export;
```
nième entrée binaire (ou entrée logique). Le résultat "vrai" correspondra à 1, et le résultat "faux" correspondra à zéro.

```
function stdneb(n:longint):pchar;stdcall;export;
```
C'est le nom de la nième entrée binaire

```
function stdsb(n:longint;valeur:longint):longint;stdcall;export;
```
fixation de la nième sortie binaire à "vrai" si valeur vaut 1, et à "faux" si valeur vaut zéro ; si tout

s'est bien passé, elle renvoie valeur

```
function stdnsb(n:longint):pchar;stdcall;export;
```
nom de la nième sortie binaire.

```
function stdtitre : pchar;stdcall;export;
```
renvoie le titre de la DLL, que l'on pourra utiliser dans des boites de dialogue.

```
function stddetail : pchar ;stdcall;export;
```
renvoie le nom détaillé de la DLL, avec par exemple le nom de l'auteur, la date de révision, etc.

2.1.2 Fonctions moins importantes, pour quelques emplois particuliers

2.1.2.a Fonction permettant d'ouvrir une fenêtre de réglage

```
function stdcalibration(pch:pchar) : pchar ;stdcall;export;
```
Cette fonction facultative provoque l'ouverture d'une boite de dialogue qui permet d'afficher et de faire les réglages. Bien sûr, le résultat est que cette DLL est plus grosse que si elle ne contenait pas de boite de dialogue. A titre d'exemple, en Delphi 5, une DLL simple fait environ 30 ko, alors qu'une DLL avec une boite de dialogue fait 250 ko.

2.1.2.b Fonctions permettant d'utiliser plusieurs logiciels d'application simultanément, avec la même interface et la même DLL

Ces fonctions ne sont utiles que pour les sorties logiques et analogiques. Ces fonctions commencent par r, comme "réponse". Il faut faire très attention lorsqu'on veut utiliser plusieurs programmes pour commander des sorties, car si chacun d'eux veut fixer la même sortie à des états différents, le résultat est souvent imprévisible ! L'emploi de ces fonctions est donc à déconseiller en général.

```
function stdrsa(n:longint):longint;stdcall;export;
```
renvoie l'état de la nième sortie analogique

```
function stdrsad(n:longint):double;stdcall;export;
```
renvoie l'état de la nième sortie analogique, convertie en volts ou autre unité appropriée.

```
function stdrsb(n:longint):longint;stdcall;export;
```
renvoie l'état de la nième sortie binaire, 0 pour hors-tension, 1 pour sous-tension.

2.1.3 Les mêmes fonctions, avec d'autres types de paramètres, pour d'autres logiciels

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

Le langage OpenBasic (StarOffice, OpenOffice, LibreOffice) permet d'appeler les DLL de façon relativement simple, mais semble exiger des DLL un peu particulières :
- Il n'accepte pas les paramètres entiers : il faut lui passer des paramètres de type «double», y compris pour indiquer les numéros de voie...
Il faut donc mettre dans la DLL des fonctions supplémentaires ayant des paramètres de type «double» ; on choisira de leur donner le même nom que les fonctions normales, mais avec le suffixe «double». Dans la pratique, peu importe que la fonction soit déclarée de type stdcall ou non, l'important est que les paramètres soient de type «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;
```

D'autres logiciels ont des difficultés avec les paramètres numériques. On peut utiliser des paramètres de type chaîne de caractères, qui sont convertis en valeurs numériques par le programme et par le pilote. Cela fait plus d'opérations informatiques, mais cela permet parfois d'utiliser d'autres logiciels.
On peut proposer le suffixe str (comme string = chaîne de caractères) :

```
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;
```

> Ces derniers types de fonctions ne sont utiles que pour quelques logiciels particuliers. Ils sont souvent absents des pilotes disponibles sur Internet, qui avaient été programmés en premier. Comme ces pilotes sont livrés avec leur programme-source, il suffit d'ajouter ces fonctions à la fin de ce programme-source, et de faire une nouvelle compilation.

3 Langages de programmation permettant de faire des bibliothèques dynamiques

Ces logiciels sont toujours des langages compilés, permettant d'obtenir des exécutables. Ils sont classés par ordre alphabétique (B, C, P), en commençant dans chaque catégorie par les logiciels libres.
Les exemples de code sont volontairement réduits, car pour faire un "vrai" pilote, il faut plusieurs dizaines de ligne de code, ce qui aurait été trop long dans ce document. L'important est de savoir qu'on peut faire des bibliothèques dynamiques avec les logiciels cités ci-dessous. Pour une réalisation pratique, le plus simple est de télécharger les exemples disponibles sur Internet (http://sciencexp.free.fr/index.php?perma=pilotes_demonstration), et de les modifier pour les adapter à votre appareil de mesure ou votre dispositif expérimental.

3.1 Langages de type "Basic"

Contrairement au Pascal (voir plus bas), l'en-tête du fichier-source ne contient pas d'indication de ce que doit être le fichier compilé. Pour que ce soit une bibliothèque dynamique, il faut l'indiquer dans des options de compilation.

3.1.1 FreeBasic (logiciel libre pour Linux et Windows)

Site internet : http://fbide.freebasic.net/, et http://sourceforge.net/projects/fbc/
Sous Linux comme sous Windows, on peut utiliser le compilateur en ligne de commande, après avoir rédigé le programme-source avec un éditeur de texte.
Sous Windows, il existe un environnement de développement intégré, qui peut rendre la rédaction des programmes beaucoup plus agréable. Cet environnement de développement peut être configuré en diverses langues par "Voir | Paramètres".
Lorsqu'on utilise la compilation en ligne de commande, il faut indiquer que la destination est une bibliothèque dynamique par "-dll" avant le nom du fichier à compiler.
Par exemple `fbc monprog.bas` va compiler le fichier monprog.bas et faire un fichier exécutable monprog.exe sous Windows et monprog sous Linux. Au contraire, `fbc -dll mabib.bas` va compiler le fichier mabib.bas pour faire une bibliothèque dynamique mabib.dll sous Windows (ou son équivalent .so sous 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
```

Sous Windows, dans l'environnement de programmation FBIDE, dans le menu "Voir | Paramètres | FreeBasic", on peut aussi indiquer au compilateur que le fichier compilé devra être une bibliothèque dynamique, là encore par le mot -dll. ("<$fbc>" -dll "<$file>")

3.1.2 PureBasic (pour Linux et Windows)

Contrairement à FreeBasic, PureBasic n'est pas un logiciel libre. Il est vendu par Fantaisie Software. Une version d'essai est téléchargeable sur internet (www.purebasic.fr, ou .com, ou .de). La version d'essai ne permet pas de compiler des bibliothèques dynamiques, mais seulement de les utiliser : pour faire des bibliothèques dynamiques, il faut la version commerciale..
Aussi bien sous Linux que sous Windows, PureBasic fonctionne par un environnement de développement intégré, en diverses langues : français, anglais, allemand ou espagnol.
Par défaut, lorsqu'on utilise l'option la plus évidente (Compilateur | Compiler-Exécuter), la compilation ne fait pas de fichier exécutable créé sur disque. Pour créer un fichier exécutable sur disque (donc ici un fichier de bibliothèque dynamique), il faut utiliser l'option Compilateur | Créer un exécutable.
Pour que la compilation soit faite sous forme de bibliothèque dynamique, il faut le préciser dans les options de compilation (Compilateur | Options du compilateur) la première fois que l'on compile le programme-source. Ensuite, ces réglages seront mémorisés dans le fichier-source, à la fin du programme lui-même (ils n'apparaissent pas dans l'éditeur de PureBasic, mais on peut les voir facilement en éditant le programme avec un autre éditeur).

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

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

3.2 Langages de type C/C++

On donne d'abord la liste des fonctions à exporter, avec leurs caractéristiques.
Ensuite, le reste du programme-source contient les fonctions en question.
Les langages C et C++ sont très puissants, et permettent de très nombreux réglages, parfois trop nombreux pour être pratiques.
Ils permettent souvent de faire deux types de compilation : "release" qui aboutit à un fichier compilé compact, mais peu analysable, et "debug", qui est moins compact, mais parfois plus utile lors de la mise au point d'un programme. Ici, il faut prendre l'option "release", car l'option "debug" aboutit souvent à un plantage du programme.
Les compilateurs C et C++ ont souvent l'habitude de "décorer" les noms des fonctions, par exemple une fonction déclarée comme "mafonction" pourra être nommée comme "__mafonction@8" (dans le fichier compilé). Pour éviter ça (pour que les noms de fonctions soient conservés dans les fichiers compilés), il y a plusieurs techniques : d'une part déclarer que ce sont des fonctions C par les mots `extern "C"` (car les fonctions C sont moins décorées que les fonctions C++), et d'autre part indiquer la correspondance des noms dans un fichier .def, où l'on trouve des lignes du type "`mafonction=mafonction`", qui signifient que la fonction "mafonction" ne doit pas changer de nom (ne pas être décorée).

3.2.1 Code::Blocks (logiciel libre, pour Linux et Windows)

(http://www.codeblocks.org)
Par défaut, cet environnement de développement utilise le compilateur gcc. Il faut choisir l'option de compilation en mode "Release".
Sous Windows, on a le choix du langage (C ou C++), mais sous Linux on doit utiliser C. En conséquence, sous Linux, on ne peut pas faire de fonctions stdcall, mais seulement des fonctions 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, pour Windows)

http://www.bloodshed.net/devcpp.html
Il utilise le compilateur libre GCC, et fournit un environnement de développement intégré.
Pour faire une bibliothèque dynamique, il faut choisir l'option Fichier | Nouveau | Projet | DLL.
Le logiciel demandera de sauvegarder le projet avec un nouveau nom (extension .dev), puis fera apparaître le squelette d'un programme-source (extension .cpp), qu'il faudra ensuite remplir avec les fonctions désirées.

```
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, pour Windows)

Pour créer une nouvelle bibliothèque dynamique, il faut prendre l'option Fichier | Nouveau | DLL. Un squelette de programme-source est créé, et il suffit ensuite de le remplir avec les fonctions souhaitées.

```
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, pour Windows)

Là aussi, il faut choisir l'option "Release", et ne pas prendre l'option "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 Langages de type Pascal

Pour réaliser une bibliothèque dynamique à la compilation, il faut que les fichiers-sources commencent par le mot "library" (alors que pour faire un programme exécutable, il faut qu'ils commencent par le mot "program").
Chaque fonction destinée à être exportée doit se terminer par "export".
A la fin de la bibliothèque, on récapitule toutes les fonctions à exporter après le mot "exports".

3.3.1 FreePascal (logiciel libre pour Linux et Windows)

http://www.freepascal.org/
Il existe plusieurs environnement de développement, soit en mode texte (FreePascal IDE) soit en mode graphique (Lazarus : http://www.lazarus.freepascal.org/). Sous Windows, on peut aussi

utiliser Bloodshed Dev-Pas (http://www.bloodshed.net/devpascal.html).
Le compilateur est très (trop?) adaptable, et on peut changer un très (trop ?) grand nombre de réglages.
Dans "FreePascal EDI", on peut effectuer ces réglages par le menu «Options Compiler».
Pour Lazarus, il faut aller dans "Projet | Options du compilateur"
Pour Dev-Pas, il faut choisir "Options | Compiler options".
Pour que FreePascal soit «vraiment» compatible avec Delphi ou TurboPascal, il faut lui en donner l'ordre en cochant les cases correspondantes dans la boite de dialogue.
De même, il y a plusieurs options pour l'utilisation d'instructions en assembleur. Apparemment, le mieux est de choisir le style «Intel».
Enfin, pour que les DLL compilées par FreePascal soient utilisables par d'autres programmes, il faut prendre le mode «normal» et non «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, pour Windows)

Ce logiciel non-libre a eu de nombreuses versions depuis 1996, qui sont toutes capables de faire des bibliothèques dynamiques et de les utiliser. L'aspect et le fonctionnement sont semblables à Lazarus.
On peut le télécharger par exemple en http://delphi.developpez.com/telecharger/gratuit/

4 Langages de programmation permettant d'appeler les bibliothèques dynamiques

Une fois que le pilote a été réalisé, on peut l'appeler assez simplement par des programmes réalisés avec divers langages de programmation.
Ces langages dont la puissance peut être augmentée par les fonctions contenues dans les bibliothèques dynamiques sont beaucoup plus nombreux que ceux qui pouvaient faire les pilotes.
Ils vont être classés dans l'ordre suivant :
- d'abord les "vrais" langages de programmation généralistes des trois grandes familles : Basic, C/C++, Pascal
- ensuite quelques autres langages généralistes : Logo, Python
- les logiciels de calcul numérique ayant des possibilités de programmation : Freemat, Scilab.
- enfin les langages de script des logiciels de bureautique.

Dans chaque catégorie seront d'abord mis les logiciels libres, puis les logiciels non libres.
Cette liste n'est pas exhaustive. Certains logiciels publiés autrefois permettaient l'utilisation de bibliothèques dynamiques, mais ne sont plus diffusés actuellement, c'est pourquoi il n'en est pas question ici. C'est le cas des logiciels Lotus (WordPro, Lotus1-2-3...), des logiciels Borland pour Linux (Kylix)... D'autres sont vendus à un prix élevé, et payer une licence uniquement pour tester la possibilité des bibliothèques dynamiques n'avait pas d'intérêt (Matlab). D'autres enfin n'ont simplement pas été testés, simplement par manque de temps.
Certains de ces langages pourront paraître puérils, d'autres trop spécialisés : l'important ici est de montrer qu'on peut utiliser les bibliothèques dynamiques, et en particulier les pilotes d'appareils de

mesure, avec des logiciels très variés. Que chacun fasse son choix !
Certains logiciels peuvent utiliser des bibliothèques dynamiques, mais qui doivent avoir leurs fonctions qui suivent une syntaxe spéciale, différente de ce système. Ils peuvent quand même utiliser ce type de pilotes, mais nécessitent un "adaptateur", qui est une bibliothèque dynamique spéciale, communiquant avec le pilote par les fonctions du pilote, et communiquant avec le logiciel par des fonctions adaptées au logiciel. C'est le cas de Scilab et des fonctions de chaîne de caractères de Freemat.
Les exemples donnés sont très simples, souvent simplement afficher à l'écran la lecture d'une entrée analogique.

4.1 Langages généralistes de type Basic

4.1.1 FreeBasic (logiciel libre, Windows et 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 (logiciel non libre, Windows et Linux)

Comme pour beaucoup de langages, il faut d'abord ouvrir la bibliothèque par OpenLibrary.
Ensuite, on peut tester l'existence d'une fonction dans la bibliothèque par GetFunction, mais ce n'est pas obligatoire.
Lorsqu'on veut utiliser une fonction de type stdcall (nom commençant par std), on l'appelle par CallFunction (Windows). Lorsqu'on utilise une fonction de type cdecl (nom commençant par c), on l'appelle par CallCFunction.
Les fonctions qui renvoient un nombre entier sont utilisables sans difficultés car CallFunction renvoie directement la valeur numérique correspondante.
L'utilisation des fonctions qui renvoient une chaîne est un peu plus délicate. L'appel de CallFunction renvoie l'adresse de la chaîne en question, et il faut aller lire cette adresse par la fonction PeekS, qui renvoie la valeur de la chaîne se trouvant à l'endroit indiqué.
Les nombres réels ont aussi besoin d'instructions particulières, avec l'emploi de "prototypes". Si la fonction doit renvoyer un nombre réel, on commence par déclarer un "prototype" d'une telle

fonction, puis on déclare une variable d'un tel type. Ensuite, on affecte à cette variable la fonction de la DLL par l'instruction getfunction.
A la fin, en principe, on ferme la bibliothèque par 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)

Ce logiciel léger (http://www.fnxbasic.com/index.html) peut appeler les bibliothèques dynamiques dont les paramètres d'entrée ou de sortie sont des entiers ou des chaînes de caractères (pas de nombres réels de type "double" !). Il lui faut apparemment des fonctions déclarées stdcall. Pour utiliser les fonctions ayant des paramètres de type "double", il faudrait faire une bibliothèque dynamique adaptatrice.
Il réalise un programme exécutable assez volumineux, mais qui fonctionne sans l'environnement de développement.

```
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 (logiciel gratuit mais non libre pour Windows)

C'est un langage Basic pour Windows (http://panoramic-language.pagesperso-orange.fr/), peu puissant sur le plan de la programmation, car il ne permet pas de déclarer des fonctions, mais seulement des sous-programmes (avec les instructions label, gosub et return), mais il permet assez facilement d'utiliser les ressources de Windows, et ses appels aux bibliothèques dynamiques tiennent en une seule ligne de programme. Il peut utiliser les bibliothèques dynamiques, mais uniquement avec des paramètres entiers. Il peut donc gérer les fonctions stdea, stdeb, stdsa, stdsb. Pour les autres fonctions, il faut utiliser une bibliothèque dynamique adaptatrice.
Lorsque le programme est au point, on peut le sauvegarder dans un fichier .exe (qui est l'interpréteur plus le programme-source).

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

4.1.5 ThinBasic (logiciel gratuit mais non libre pour Windows)

Ce langage interprété est disponible en http://www.thinbasic.com/. Il dispose d'un grand nombre de fonctions par l'utilisation systématique de "modules", qui sont des bibliothèques dynamiques. Il peut utiliser aussi les bibliothèques du système Mensurasoft.

```
PrintL "bonjour, "
PrintL stdead(0)
Sleep(2000)
PrintL stdead(0)
PrintL mondetail()
PrintL monnead(0)
PrintL "appuyez sur q"
Do
 Loop Until Console_InKey  ="q"
```

4.1.6 Gambas (logiciel libre pour Linux)

C'est un langage Basic pour Linux, disponible en http://gambas.sourceforge.net/fr/main.html.

Il peut appeler les bibliothèques dynamiques du système Mensurasoft.

Exemple :

```
' 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 Langages généralistes de type C ou C++

Les mini-programmes ci-dessous sont le contenu des fichiers main.cpp générés par les logiciels en question.

4.2.1 Code::Blocks (logiciel libre, Linux et 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, pour 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 Langages généralistes de type Pascal

4.3.1 FreePascal (logiciel libre, Windows et Linux)

4.3.1.a Editeur 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.b Environnement 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 Autres langages généralistes : Logo, Python

4.4.1 Langage Logo (MSW-Logo)

(http://mswlogo.softonic.fr/telecharger)
C'est très simple :
- On charge la DLL par l'instruction dllload :

```
dllload "xadestar.dll
```

- puis on utilise les fonctions par l'instruction dllcall, à laquelle on passe en paramètre le nom de la fonction (et éventuellement des paramètres) :

```
print dllcall[s stddetail]
print dllcall[f stdead l 0]
```

où f indique le type du résultat renvoyé par la fonction («float», c'est à dire un nombre réel de type double), w le type du paramètre («word», c'est à dire un entier), et 0 est la valeur de ce paramètre
- enfin, on libère la mémoire de la DLL par

```
dllfree
```

Symboles des types de paramètres et de résultats :
v = void (rien)
w = word (entier)
l = longword (entier long)
f = float (réel de type double)
s = string (spstr = chaîne de caractères à zéro terminal)

Quelques remarques :
- l'état majuscule/minuscule est important
- apparemment, pour les fonctions avec paramètres, il faut utiliser les fonctions déclarées avec le mot stdcall, ou à paramètres de type double.
- apparemment aussi, l'ordre des paramètres est inversé par rapport à la déclaration en Delphi. Pour les fonctions avec deux paramètres, il faut appeler d'abord le deuxième, puis le premier.
- enfin, il semble qu'une seule DLL est chargeable à la fois. Pour en utiliser une deuxième, il faut

d'abord décharger la première.

En résumé, pour faire une fonction «essai» qui affiche le nom de l'appareil, le nom de l'entrée analogique 0 et la valeur de cette entrée 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 l 0]
dllfree
end
```

puis de sauvegarder ceci, et de frapper «essai» dans la ligne de commande.
Voir sur Internet : http://www.educa.fmf.uni-lj.si/logo/eurologo.01/paper1.doc

4.4.2 Langage Python

téléchargeable en : http://www.python.org/

Python est un langage interprété, existant pour Windows comme pour Linux (et d'autres systèmes d'exploitation).
Les fonctions d'importation de bibliothèques dynamiques sont dans ctypes. "windll" permet de charger les fonctions de type stdcall, alors que "cdll" permet de charger les fonctions cdecl.
Dans windll ou dans cdll, LoadLibrary permet de charger la bibliothèque dynamique, avec toutes ses fonctions disponibles. On peut ensuite accéder à ces fonctions par leur nom (séparé par un point).
Il n'y a pas de problème pour les fonctions travaillant avec des entiers (comme stdea, cea, stdeb, ceb...). Par contre, pour les fonctions renvoyant un réel, ou celles renvoyant un pointeur de chaîne, il faut l'indiquer par 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 "fini !"
```

4.5 Logiciels de calcul : Freemat, Scilab

4.5.1 Freemat (logiciel libre pour Linux et Windows, mais la version Linux semble ne pas gérer les bibliothèques dynamiques)

Téléchargeable en : http://freemat.sourceforge.net/
La syntaxe est en principe assez simple.

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

Ensuite, on peut utiliser la fonction avec son nom (le 3e paramètre).
Freemat ne fonctionne directement qu'avec les fonctions numériques de ce système de pilotes (cead, csad, ceb, csb), mais non avec les fonctions renvoyant une chaîne de caractères. Pour utiliser toutes les fonctions du système (cdetail, ctitre, cnead, cnsad, cneb, cnsb), il faut réaliser une adaptation par une bibliothèque dynamique spéciale ("adaptateur").
La version 3.6 de Freemat accepte les appels de paramètres de type cdecl et stdcall, mais la version 4 n'accepte que les fonctions de type cdecl.

4.5.2 Scilab (logiciel libre pour Linux et Windows)

Téléchargeable en : http://scilab.org

Scilab nécessite une syntaxe particulière pour ses bibliothèques dynamiques, différente de celle de ce système de pilotes. Pour que Scilab gère ce système, il lui faut un adaptateur sous forme d'une bibliothèque dynamique spéciale, qui gère les fonctions des pilotes et communique avec Scilab selon la syntaxe de Scilab.
Une fois ceci réalisé, l'ensemble fonctionne correctement, sous Linux comme sous Windows.

4.6 Langages de script des logiciels de bureautique

4.6.1 OpenBasic et LibreBasic, pour OpenOffice et LibreOffice (logiciels libres, Linux et Windows, mais la version Linux ne semble pas gérer les bibliothèques dynamiques)

Au début du module, on déclare les fonctions à utiliser dans les DLL :

```
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
```

Ensuite, dans le corps du programme en Basic, on utilise ces fonctions dans un sous programme, appelé par exemple «mesures» . Le sous programme ci-dessous va insérer la valeur lue à la voie 1 dans la case B1 de la première feuille de calcul (nommée «Feuille1») :

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

Enfin, il faut donner l'ordre de déclencher ce sous-programme, par exemple en créant un bouton, et en spécifiant que lors du déclenchement de ce bouton, le sous-programme «mesures» sera effectué.

5 Où trouver des logiciels, des pilotes, et des exemples de programmation ?

Le site http://sciencexp.free.fr propose diverses ressources :

5.1 Des pilotes d'appareils de mesure, avec leur programme-source

On peut y trouver de vrais pilotes d'appareils de mesure :

- interfaces d'acquisition scientifiques Orphy (Micrelec), ESAO (Jeulin), Candibus, PMB, Cassy (Leybold), Eurosmart...

- interfaces d'expérimentation : Arduino, Velleman K8055, ExpEyes

- appareils de mesure spécialisés, à connecter sur un port série, parallèle ou USB : pHmètres, thermomètres, oxymètres, spectrophotomètres...

- bricolages divers à connecter sur une prise USB, parallèle, ou une prise joystick

On y trouve aussi des pilotes de démonstration, qui renvoient des mesures fictives ou bien des valeurs provenant de l'horloge interne de l'ordinateur, nommés les "pilotes-système". Leur intérêt principal est d'être fournis sous forme de programme-source, en Basic (FreeBasic et PureBasic), C (Code::Blocks, Borland C++Builder ...) ou Pascal.(FreePascal, Lazarus ou Delphi). Pour les adapter à de vrais appareils électroniques, il suffit de changer quelques lignes dans les fonctions déjà existantes, et de recompiler le programme pour obtenir la bibliothèque dynamique.

5.2 Des programmes d'application, avec leur programme-source

5.2.1 Mensurasoft-LZ, logiciel libre pour Windows et Linux

LZ signifie qu'il a été programmé en langage Lazarus/FreePascal. Le principe est le même que Mensurasoft-PB et MGW32 : acquisition possible sur 3 voies, correspondant éventuellement à 3 pilotes différents de 3 appareils différents, avec gestion de sortie analogique programmable pendant l'acquisition d'une série de données, et gestion de sorties binaires et analogiques en dehors de l'acquisition d'une série de données. On peut l'utiliser avec des menus et boites de dialogue en diverses langues, par des fichiers de langues facilement modifiables. Les résultats des mesures peuvent être exportés soit par le presse-papier, soit par des fichiers lisibles par des tableurs.

5.2.2 Mensurasoft-PB, logiciel libre pour Windows et Linux

C'est un logiciel pouvant faire des mesures sur 3 voies, éventuellement sur 3 appareils différents, grâce à 3 pilotes différents. Il peut aussi commander les sorties logiques et analogiques. Les données peuvent ensuite être copiées vers le presse-papier ou bien enregistrées dans des fichiers-textes lisibles par les tableurs-grapheurs.

C'est un logiciel multilingue grâce à des fichiers de langages.

5.2.3 MGW32 pour Windows

Ce logiciel est plus ancien que Mensurasoft-PB, et il ne fonctionne que sous Windows, mais il est un peu plus puissant.

Il a les mêmes caractéristiques que Mensurasoft-PB, mais travaille en mode multifenêtre, ce qui permet de visualiser plusieurs séries de mesures simultanément.

5.2.4 Des exemples de programmes dans différents langages, pour Linux et Windows

Ces programmes sont très simples : en général ils permettent simplement de charger un pilote, et d'afficher les résultats de la lecture sur une ou quelques voies.

Leur intérêt est de fournir des programmes-sources pour différents langages de programmation, ce qui peut servir de base pour la réalisation de logiciels plus complexes.

- Basic : FreeBasic (Linux et Windows), PureBasic (Linux et Windows), FNXBasic (Windows), Panoramic (Windows), Gambas (Linux)

- C/C++ : Code::Blocks (Linux et Windows), Visual C++ (Windows)

- Pascal (FreePascal et Lazarus pour Linux et Windows, Delphi pour Windows)

- Python (Linux et Windows)

- MSW Logo (Windows)

- logiciels de calcul numérique Freemat et Scilab

- logiciels de bureautique OpenOffice et LibreOffice (uniquement pour Windows, car les versions pour Linux ne semblent pas pouvoir utiliser les bibliothèques dynamiques).

Index lexical