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

# Emploi de bibliothèques dynamiques comme pilotes pour les logiciels d'EXAO (Windows 32 bits et Linux)

On a souvent coutume d'opposer Windows, le système d'exploitation commercial de Microsoft, à Linux, le système d'exploitation gratuit.
En fait, un certain nombre de recettes sont utilisables dans les deux environnements, et l'on peut développer des logiciels pour l'un ou pour l'autre de ces systèmes.

## 1 Principe fondamental : utiliser des "pilotes d'appareils de mesure", spécifiques de l'appareil utilisé, pour faire des logiciels généralistes.

### *1.1 Très (trop ?) grande diversité du matériel*

Il y a une grande diversité dans le matériel de mesure connectable à un ordinateur : certains appareils sont des cartes à enficher 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...) 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.

Dans les explications qui suivent, il sera surtout question du boitier ADES, diffusé par la Maison des Enseignants de Provence. C'est un petit boitier à brancher sur la prise parallèle (prise "imprimante"), et qui a deux entrées analogiques sur 10 bits (normalement entre 0 et 5 V) et 4 sorties logiques sous forme de relais. Des adaptateurs lui permettent d'utiliser les capteurs de Calibration, fondamentalement conçus pour être utilisés avec les calculettes de la série TI82 (mesure d'oxygène, de pH, etc).
Il sera aussi question de l'interface Orphy-GTS, diffusée par Micrelec, et qui possède un grand nombre d'entrées et de sorties logiques et analogiques. Cet appareil doit être connecté sur une prise série.
A partir de ces deux appareils, on peut imaginer que les solutions envisagées sont universelles.

### *1.2 Plusieurs systèmes de pilotes sont possibles*

#### 1.2.1 Bibliothèques dynamiques (« DLL » de Windows, ou « so » de Linux)

Ce sont des petits fichiers contenant des instructions spécifiques, et qui sont appelables directement par le logiciel principal. Lorsque le fichier a été chargé par le logiciel principal (« lié au logiciel »), celui-ci peut utiliser les fonctions de la bibliothèque

comme si elles faisaient directement partie du logiciel ou du langage de programmation. Ceci permet donc des mesures très rapides (avec un intervalle de l'ordre de la microseconde).
On peut réaliser de telles bibliothèques dynamiques au moyen des langages compilés de type Pascal-(Delphi, Kylix, FreePascal), des divers types de C++, ou de certains langages Basic compilés (PureBasic, FreeBasic). On peut utiliser ces bibliothèques dynamiques par les mêmes langages, ainsi que par d'autres langages ou logiciels : OpenOffice (au moins sous Windows), Freemat, MSWLogo, Python...
Malheureusement, tous les langages de programmation ne permettent pas l'appel de ces bibliothèques dynamiques.

### 1.2.2 Programmes-pilotes faisant de petits fichiers

On peut réaliser de petits programmes qui font les mesures, et mettent la valeur mesurée dans un petit fichier. Le logiciel principal lit ce fichier, dont la valeur peut changer au cours du temps, et peut alors tracer le graphique de la valeur en fonction du temps, faire des calculs statistiques, etc.
Cette méthode est plus universelle que la méthode des bibliothèques dynamiques, puisqu'elle est possible même pour des langages de programmation qui ne permettent pas de faire de telles bibliothèques, mais elle est beaucoup plus lente, et il ne faut pas vouloir faire de mesures plus rapidement qu'une ou quelques fois par seconde.
Deux variantes en sont possibles :

- Un programme-pilote indépendant, que l'on lance d'abord, et qui fait ses mesures et ses fichiers à intervalles réguliers. Il reste en fonctionnement pendant toute la durée de la série de mesures.
- Un programme-pilote qui est lancé sur ordre du logiciel principal, qui règle donc la fréquence des mesures. Le programme-pilote fait sa mesure, en inscrit le résultat dans un fichier, puis se termine.

### 1.2.3 Programmes-pilotes renvoyant une chaîne de caractères

Fondamentalement, lorsqu'on lance ces programmes-pilotes, ils font une mesure, et affichent le résultat à l'écran, par une instruction du type « writeln(mamesure) » de Pascal, ou « print(mamesure) » de Basic.
Leur intérêt est que certains logiciels (par exemple Scilab) peuvent lancer de tels programmes extérieurs et récupérer la chaîne de caractères.

*La suite de ce texte ne décrira que les pilotes de type « bibliothèque dynamique ». Les exemples seront d'abord décrits pour le système Microsoft-Windows, puis pour le système Linux.*

# 2 DLL ("Dynamic linked library" = librairies à liaison dynamique) de Windows

## *2.1 Principe*

Dans le système proposé ici, chaque DLL de mesure sera caractéristique d'un appareil de mesure (ou d'une façon d'utiliser cet appareil). La DLL contiendra les fonctions de communication avec l'appareil de mesure : lecture des entrées analogiques et logiques, commande des sorties logiques et analogiques, ainsi que des fonctions de service : indication du nom de l'appareil, de la version du pilote, du nom et du nombre des voies d'entrée et de sortie, de l'état des sorties, etc. C'est la DLL qui est capable d'envoyer les bons signaux sur la voie RS232, ou bien de changer l'état des ports de l'ordinateur... Par contre, la DLL ne fera pas les travaux compliqués tels que le traçage des graphiques des points de mesure, ou bien la sauvegarde des résultats dans un fichier compatible avec les tableurs...

Le logiciel d'application lui-même appellera les fonctions de la DLL (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 à la DLL 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. Ce logiciel est développé en Delphi 1, qui est un langage de type Pascal, conçu pour les versions 16 bits de Windows (Windows_3.1). Il fonctionne 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.

## 2.2 Où trouver des renseignements sur les bibliothèques dynamiques ?

Malheureusement, les livres de programmation documentent rarement les fonctions permettant de créer ou d'utiliser des bibliothèques de fonctions extérieures à chargement dynamique.
Un bon site internet sur ces bibliothèques dynamiques, aussi bien pour Windows que pour Linux, en allemand, est :
`http://www.wachtler.de/dynamischeBibliotheken/dynamischeBibliotheken.html`

## 2.3 Problèmes des «conventions d'appel» et des types de variables

### 2.3.1 Les conventions d'appel

Dans le détail, il y a quelques complications, en particulier à cause du passage des paramètres.
Selon la façon dont l'échange est fait entre le programme principal et la librairie, on peut distinguer plusieurs possibilités :

| Directive en C++Builder | Directive en Delphi-Pascal | Ordre des paramètres | Nettoyage | Transfert de paramètre dans les registres | transformation du nom de la fonction «mafonction» par C++Builder |
|---|---|---|---|---|---|
| __fastcall | register (par défaut) | De gauche à droite | Routine | Oui | @mafonction |
| pascal ou __pascal | pascal | De gauche à droite | Routine | Non | pascal : mafonction __pascal : MAFONCTION |
| cdecl ou __cdecl (par défaut) | cdecl | De droite à gauche | Appelant | Non | cdecl : @mafonction __cdecl : _mafonction |
| __stdcall | stdcall | De droite à gauche | Routine | Non | mafonction |
| | safecall | De droite à gauche | Routine | Non | (pour des «méthodes d'interface dual», sans intérêt ici |

Il y a donc un double problème :
D'une part, il faut que la convention d'appel soit la même pour la librairie et le programme appelant.
D'autre part, il faut qu'il n'y ait pas de problèmes de nom, c'est à dire que le programme appelant trouve bien la fonction avec le nom qu'il cherche. Or C++Builder change souvent le nom des fonctions ! Si, dans le texte-source, on déclare une fonction avec le nom mafonction (par la syntaxe `int __fastcall mafonction(int n)` ), en fait, le programme écrit en Delphi devra chercher la fonction `@mafonction` !
C'est une source d'embrouilles multiples.
Par conséquent, le mieux est de choisir d'utiliser une convention d'appel possible à la fois en Delphi et en C++, et qui ne pose pas de problème de nommage. Le mieux est de choisir par défaut la convention «stdcall», d'autant plus que std signifie «standard», et que la standardisation est justement notre propos. Il sera aussi possible d'utiliser d'autres conventions, mais en donnant un nom différent aux fonctions, pour éviter les confusions.

### 2.3.2 Types de variables

Il est nécessaire que le programme principal et la bibliothèque dynamique aient le même type de données lorsqu'ils échangent des informations.
Dans les différents langages de programmation, il existe une multitude de types entiers : codés sur un, deux, quatre octets ou même davantage, devant être positifs ou bien pouvant être aussi négatifs... Le type le plus universel est celui codé sur 4 octets, signé (c'est à dire pouvant être positif ou négatif), nommé « longint » dans les langages Pascal, « long » en PureBasic et OpenBasic, integer en Python et FreeBasic.
Parmi les types « réels », c'est à dire décimaux, le type le plus universel est celui codé sur 8 octets, nommé « double » en Pascal, PureBasic, FreeBasic et OpenBasic, et « float » en Python.
Enfin, les chaînes de caractères peuvent aussi être codées de diverses façons. On choisira le type « chaîne à zéro terminal » des langages Pascal, qui est le type de base du langage C++.

Les booléens seront représentés par des valeurs entières : 0 pour « faux » (false), et 1 pour « vrai » (true), comme dans bon nombre de langages, en particulier les dialectes de Basic.

# 3 Syntaxe proposée pour les DLL :

Ce paragraphe correspond à une explication du "pourquoi et comment" a été fait le choix des caractéristiques des fonctions proposées. La syntaxe indiquée ici correspond au langage Pascal (et Delphi) ; des exemples sont donnés plus loin avec le langage C++ et le langage Basic.

## *3.1 Fonctions de base*

La numérotation des voies commencera à zéro, au moins pour ce qui est de la programmation, le paramètre correspondant sera donc un entier positif (mais désigné par un type « longint », pour employer le moins de types possible). Ceci n'empêche pas éventuellement que la première voie (numérotée 0 pour la programmation) puisse porter un nom tel que "Voltmètre 1".

Si la fonction renvoie une valeur de type "réel" (en virgule flottante), ce type sera codé sur 8 octets (type "double", qui est le type de réels le plus répandu sur les ordinateurs actuels.

Si la fonction renvoie une valeur de type entier, le mieux est que ce soit un type entier signé sur 4 octets (« longint »), même si normalement un convertisseur analogique-numérique donne une valeur entière positive, parce que les nombres négatifs peuvent être utilisés pour indiquer qu'il y a eu un problème dans la conversion.

Si la fonction a besoin d'autres paramètres de type entier (sorties analogiques ou sorties logiques), ces paramètres seront aussi de type « entier long » (« longint », codé sur 4 octets.

La syntaxe proposée ici est voisine de celle commandant la centrale de mesure Orphy-GTS de Micrelec ; si l'opération qu'elles doivent faire est incorrecte (voie inexistante, par exemple) elles renvoient une valeur fixe (par exemple -777 pour ce qui est des fonctions numériques). Pour qu'on sache bien qu'il faut utiliser la convention d'appel stdcall, le nom des fonctions commence par std.

Bien sûr, il est possible de rajouter des fonctions d'autres types, par exemple, pascalea, pascalnea, pascalead, pascalnead ... pour les fonctions de type "pascal", ou cea, cnea, cead, cnead... pour les fonctions de type cdecl, etc. Ces fonctions d'autres types devraient être peu utiles, puisque la plupart des logiciels capables d'utiliser les bibliothèques dynamiques peuvent utiliser les fonctions avec la convention d'appel stdcall.

Pour les réels ( de type "double"), la fonction sera précédée de p (comme "Pointeur") lorsqu'elle renverra l'adresse du résultat et non celui-ci directement.

Lors du chargement de la DLL, chaque fonction est identifiée par son nom, qui est une chaîne de caractères. Puisqu'il y a une différence entre les majuscules et les minuscules ; il vaut mieux convenir que tous les noms de fonctions seront en minuscule.

Chaque variante de fonction de mesure sera associée à un nom particulier.
Un premier intérêt en est d'indiquer clairement à quoi correspond la variante. Par exemple, pour les interfaces ayant plusieurs calibres sélectionnables logiciellement (Leybold-Cassy...), chaque voie de mesure correspondra à un calibre, qui sera indiqué dans ce nom, ou bien pour les interfaces ayant plusieurs voies, spécialisées chacune dans un type de mesure (Orphy-GTS, qui a des calibres différents selon les connecteurs, ou bien Pierron SMF10, qui a des entrées spécialisées pour le pH ou le magnétisme...), ce nom indiquera ces spécialités.
Un deuxième avantage, plus universel, est que ce nom permettra au programme appelant de savoir quelles sont les voies réellement appelables. Toutes les voies ayant un nom (non nul) seront appelables, et les voies qui n'existent pas réellement auront un nom de longueur nulle, ce qui permettra au logiciel appelant de savoir que ces voies n'existent pas.

Le plus simple est de faire commencer le nom de la fonction qui renvoie le nom de la voie par la lettre , comme "nom". Le format de ces nom sera le type "chaîne à zéro terminal" ou "Asciiz", déclaré dans le langage Pascal comme Pchar ; ce format est le format de chaînes standard dans les langages C et C++.

```
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.

## *3.2 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, de façon à mémoriser leur état dans la DLL elle-même, pour qu'une même DLL puisse être appelée par plusieurs logiciels différents.
Ces fonctions commencent par r, comme "réponse" :

`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.

## *3.3 Fonction permettant d'ouvrir une fenêtre de réglage*

`function stdcalibration(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.

Par exemple pour Mesugraf pour Windows 16 bits, il a été fait une DLL pour ADES et une sonde à oxygène. Pour que la valeur renvoyée soit effectivement une concentration en oxygène, il faut calibrer cette sonde, en indiquant les valeurs correspondant au zéro en oxygène, et à la teneur atmosphérique en oxygène. Cette fenêtre de calibration s'obtient par l'appel à la fonction «calibration».

## *3.4 Fonctions permettant d'utiliser des logiciels plus variés*

Les fonctions précédentes ont été définies pour faire de la programmation avec Delphi, aussi bien pour la programmation des DLL que pour la programmation des logiciels d'application. Elles sont suffisantes pour ce but.
Malheureusement, tous les logiciels (tous les langages de programmation) n'acceptent pas d'utiliser ces DLL aussi simples, et demandent diverses adaptations.

### 3.4.1 Fonctions renvoyant des pointeurs de doubles

Les tableurs sont des logiciels spécialisés dans la représentation graphique et les calculs sur des séries de données. Il est tentant de faire arriver directement les mesures dans les cases des tableurs.

Vers 1992, Excel 5 pouvait faire les mesures en utilisant la fonction «fonction.appelante». Tout se passe bien tant qu'on n'utilise que les fonctions entières, mais pour les fonctions devant renvoyer une valeur réelle (de type «double»), les fonctions ne conviennent pas. Il fallait à Excel 5 des fonctions qui renvoient un «pointeur de double».

```
type pdouble = ^double;

function pead(n:word):pdouble ;export;
```

pointeur sur la valeur renvoyée par la nième entrée analogique.

```
function psad(n:word; valeur:pdouble):pdouble;export;
```

Pointeur ayant comme paramètre un pointeur sur la valeur à fixer.

Est-ce encore possible avec les versions récentes de Excel ?

### 3.4.2 Paramètres de type « double », pour utilisation avec la suite bureautique Star Office et OpenOffice

Le langage StarBasic = OpenBasic 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;
```

### 3.4.3 Autres langages de programmation, autres problèmes

Il n'est pas possible de tester tous les langages. Le langage Borland C++Builder accepte aussi d'utiliser les DLL réalisées en Delphi, mais ce langage fait une distinction entre les majuscules et les minuscules, contrairement à Delphi.

Visual Basic semble exiger la déclaration des fonctions avec l'option stdcall, ainsi que MSWLogo.

# 4 Exemples de bibliothèques dynamiques programmées en Pascal, C++ et Basic

## *4.1 Langages Pascal*

### 4.1.1 exemple en Pascal (Delphi 5) : DLL pour ADES connecté à la prise imprimante (1e prise parallèle)

```
library xadestar;

uses
  adesunit,sysutils,wintypes;

type array100= array[0..100] of char;
type pdouble=^double;
var memosb:array[0..3]of boolean;
    memosad:double;

function stdea(n:longint):longint;stdcall;export;
var o,n1,n2:word;
begin
  o:=n;
  stdea:=-777;
  if (n=0)or(n=1)then lit_niveau(n1,n2);
  case o of 0: stdea:=n1;
            1: stdea:=n2;
            2: ades_off;
            3: ades_on;
            end;
end;

function stdnea(n:longint):pchar;stdcall;export;
begin
if (n=0)
    then stdnea:='E. analogique 0'+chr(0)
      else if n=1 then stdnea:='E. analogique 1'+chr(0)
      else if n=2 then stdnea:='RAZ pour impression'+chr(0)
             else if n=3 then stdnea:='Réinitialisation'+chr(0)
    else stdnea:=chr(0);
end;

function stdead(n:longint):double;stdcall;export;
var aux:integer;
begin
  if (n=0) or (n=1)or (n=2) or (n=3)
     then begin aux:=stdea(n); stdead:=(5.0*aux)/1024; end
     else stdead:=-777;
end;

function stdnead(n:longint):pchar;stdcall;export;
begin
  if (n=0)
    then stdnead:='E. analogique 0'+chr(0)
      else if n=1 then stdnead:='E. analogique 1'+chr(0)
      else if n=2 then stdnead:='RAZ pour impression'+chr(0)
             else if n=3 then stdnead:='Réinitialisation'+chr(0)
    else stdnead:=chr(0);
end;

function stdsa(n:longint; valeur:longint):longint;stdcall;export;
begin
 stdsa:=-777; (*Ades n'a pas de sortie analogique*)
end;

function stdnsa(n:longint):pchar;stdcall;export;
begin
```

```
if n=0 then  stdnsa:='SA fictive'+chr(0) else stdnsa:=''+chr(0);
end;

function stdsad(n:longint ; valeur:double):double;stdcall;export;
begin
MEMOsad:=valeur; stdsad:=0;
end;

function stdnsad(n:longint):pchar;stdcall;export;
begin
   if n=0 then stdnsad:='SA fictive'+chr(0)else stdnsad:=''+chr(0);
end;

function stdrsa(n:longint):longint;stdcall;export;
begin       (*état de la sortie analogique fictive*)
if n=0 then stdrsa:=2 else stdrsa:=-777;
end;

function stdrsad(n:longint):double;stdcall;export;
begin
  if n=0 then stdrsad:=2 else stdrsad:=-777;
end;


function stdeb(n:longint):longint;stdcall;export;
begin  (*entrées logiques fictives, inexistantes sur ADES*)
 if n=0 then stdeb:=0 else if n=1 then stdeb:=1 else stdeb:=-777;
end;

function stdneb(n:longint):pchar;stdcall;export;
begin
 if n=0 then stdneb:='EB fictive 0'+chr(0)
        else if n=1 then stdneb:='EB fictive 1'+chr(0)
                    else stdneb:=chr(0);
end;

function stdsb(n:longint; etat:longint):longint;stdcall;export;
begin
 case n of
   0..3 : begin relais[n+1]:=(etat=1);
   stdsb:=n*100+etat;
   pilote_relais(0);
   memosb[n]:=(etat=1);
          end;
   else stdsb:=-777;
    end;
end;

function stdnsb(n:longint):pchar;stdcall;export;
begin
result:=chr(0);
if n=0 then result:='relais A'+chr(0);
if n=1 then result:='relais B'+chr(0);
if n=2 then result:='relais C'+ chr(0);
if n=3 then result:='relais D'+chr(0);
end;

function stdrsb(n:longint):longint;stdcall;export;
begin
  if n in [0..3] then if memosb[n] then stdrsb:=1 else stdrsb:=0
                 else stdrsb:=-777;
end;

function stdtitre : pchar;stdcall;export;
begin stdtitre:='ADES en LPT1:'+chr(0);end;

function detail : pchar;
begin stddetail:='essai de DLL (32 bits) pour ADES 10 bits en LPT1:, par P.
Dieumegard, le 10 août 2002'+chr(0);end;
```

```
(*les deux fonctions qui suivent sont pour EXCEL ???*)
function peaD(n:longint):pdouble;export;
var dloc:double;
begin dloc:=eaD(n);pead:=@dloc;end;

function PSAD(n:longint;valeur:pdouble):pdouble;export;
var dloc1,dloc2:double;
begin dloc1:=valeur^;dloc2:=SAD(n,dloc1);psad:=@dloc2;end;

(*les fonctions " double " qui suivent sont pour StarOffice ;
en fait, il est possible que les fonctions stdxxx soient aussi possibles*)
function eadouble(x:double):double;
var nloc:integer;
begin  nloc:=round(x);  eadouble:=ead(nloc);end;

function neadouble(x:double):pchar;
begin
  neadouble:=nead(round(x));
end;

function sadouble(x:double;xval:double):double;
begin sadouble:=sad(round(x),xval);end;

function nsadouble(x:double):pchar;
begin nsadouble:=nsad(round(x));end;

function rsadouble(x:double):double;
begin rsadouble:=rsad(round(x));end;

function ebdouble(x:double):double;
begin ebdouble:=eb(round(x));end;

function nebdouble(x:double):pchar;
begin nebdouble:=neb(round(x));end;

function sbdouble(n:double;etat:double):double;
begin sbdouble:=sb(round(n),round(etat));end;

function nsbdouble(n:double):pchar;
begin nsbdouble:=nsb(round(n));end;

function rsbdouble(n:double):double;
begin rsbdouble:=rsb(round(n));end;

exports
  pead ,
  psad ,

 eadouble ,
  neadouble ,
  sadouble,nsadouble,rsadouble,
  ebdouble,nebdouble,
  sbdouble, nsbdouble,rsbdouble,
  stdea, stdnea,stdsa,stdnsad,stdtitre,stddetail,
  stdead,stdnead,stdsad,stdnsad,stdeb,stdneb,stdsb,stdnsb,stdrsad,stdrsb;

var i:word;
begin
 nbitades:=10;
 initialise_ades(888);
 ades_on;
 for i:=0 to 3 do memosb[i]:=false;
end.
```

### 4.1.2 Utilisation de FreePascal pour écrire une DLL

FreePascal est un langage pascal gratuit, existant pour plusieurs systèmes d'exploitation, en particulier MS-Windows et Linux. Comme Delphi, il peut faire les DLL et les utiliser.
Le compilateur est très (trop?) adaptable, et on peut changer un très (trop ?) grand nombre de réglages.

Une information très utile pour éviter du travail inutile : 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 «Options Compiler».

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 Delphi, il faut prendre le mode «normal» et non «debug».

Finalement, la bibliothèque initialement faite pour Delphi 5 est compilable par FreePascal avec peu de modifications, et le résultat est utilisable par les logiciels écrits en Delphi.

## 4.2 Une bibliothèque pour un appareil fictif, en C++Builder 4

Dans cette bibliothèque, initialement nommée xdllvide_cpp.cpp, il n'y a que les fonctions «stdcall» : comme on l'a vu précédemment, des problèmes peuvent se poser dans la transmission des paramètres, et le type «stdcall» semble le plus répandu. sous Windows.  C'est donc lui qui a été choisi.

```
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
        return 1;
}
//---------------------------------------------------------------------------

int memosa;
double memosad;
bool memosb[4];

typedef unsigned char * tpchar;
//pour avoir la correspondance avec le type pchar de Delphi
//Avec Linux, il faut supprimer le «unsigned», mais en C++Builder,
//cela fonctionne quand même


extern "C" __declspec(dllexport) __stdcall signed short stdea(unsigned short n);
extern "C" __declspec(dllexport)  __stdcall tpchar stdnea(unsigned short n);
extern "C" __declspec(dllexport) __stdcall double stdead(unsigned short n);
extern "C" __declspec(dllexport) __stdcall tpchar stdnead(unsigned short n);
extern "C" __declspec(dllexport) __stdcall signed short stdsa(unsigned short n,unsigned short
val);
extern "C" __declspec(dllexport) __stdcall tpchar stdnsa(unsigned short n);
extern "C" __declspec(dllexport) __stdcall double stdsad(unsigned short n, double val);
extern "C" __declspec(dllexport) __stdcall tpchar stdnsad(unsigned short n);
extern "C" __declspec(dllexport) __stdcall signed short stdrsa(unsigned short n);
extern "C" __declspec(dllexport) __stdcall double stdrsad(unsigned short n);
extern "C" __declspec(dllexport) __stdcall signed short stdeb(unsigned short n);
extern "C" __declspec(dllexport) __stdcall tpchar stdneb(unsigned short n);
extern "C" __declspec(dllexport) __stdcall signed short stdsb(unsigned short n , unsigned
short etat);
extern "C" __declspec(dllexport) __stdcall tpchar stdnsb(unsigned short n);
extern "C" __declspec(dllexport) __stdcall signed short stdrsb(unsigned short n);
extern "C" __declspec(dllexport) __stdcall tpchar stddetail(void);
extern "C" __declspec(dllexport) __stdcall tpchar stdtitre(void);

//---------------------------------------------------------------------------
signed short  __stdcall stdea(unsigned short n)
{ if (n>-1 && n<3) {return n*3;}else{return -777;} }

tpchar  __stdcall stdnea(unsigned short n)
{
tpchar chloc;
 chloc="\0";
 if (n==0) chloc="entrée analogique 0\0";
 if (n==1) chloc="EA 1 (volts)\0";
 if (n==2) chloc="température °C\0";
   return chloc; }

double  __stdcall stdead(unsigned short n)
```

```
{
double varloc;
if (n>-1&&n<3) {varloc= n*3.33;}else{varloc= -777;}
return varloc;
 }

tpchar __stdcall stdnead(unsigned short n)
{
tpchar chloc;
 chloc="\0";
 if (n==0) chloc="entrée analogique 0\0";
 if (n==1) chloc="EA 1 (volts)\0";
 if (n==2) chloc="température °C\0";
   return chloc;  }

signed short  __stdcall stdsa(unsigned short n , unsigned short val)
{ if (n==0) {memosa=val ;return val;}else{return -777;}
}

tpchar  __stdcall stdnsa(unsigned short n)
{ tpchar chloc;
if (n==0) chloc="sortie analogique\0"; else chloc="\0";
return chloc; }


double  __stdcall stdsad(unsigned short n, double val)
{ if (n==0){memosad=val; return val;} else return -777;
}

tpchar  __stdcall stdnsad(unsigned short n)
{if (n==0) return "SA 0 (volts)\0"; else return "\0"; }

signed short  __stdcall stdrsa(unsigned short n)
{ if (n==0) return memosa; else return -777; }

double __stdcall stdrsad(unsigned short n)
{ if(n==0) return memosad; else return -777;}

signed short  __stdcall stdeb(unsigned short n)
{
if (n==0)
        return 0;
        else if (n==1)
                return 1;
                else return -777;
}

tpchar  __stdcall stdneb(unsigned short n)
{ if (n==0)
        return "EB fictive 0\0";
        else if (n==1)
                return "EB fictive 1\0";
                else return "\0";
}

signed short  __stdcall stdsb(unsigned short n, unsigned short etat)
{ int varloc;
  varloc=-777;
//mettre ici la commande des sorties binaires,
// ainsi que la mémorisation dans le tableau memosb;
  memosb[n]=etat;
  return etat;
  }

tpchar  __stdcall stdnsb(unsigned short n)
{tpchar varloc;
varloc=="\0";
if (n==0) varloc="Relais 0\0";
if (n==1) varloc="Relais 1\0";
if (n==2) varloc="Relais 2\0";
if (n==3) varloc="Relais 3\0";
return varloc;  }

signed short  __stdcall stdrsb(unsigned short n)
{int varloc;
varloc=-777;
if (n==0) varloc=memosb[0];
```

```
if (n==1) varloc=memosb[1];
if (n==2) varloc=memosb[2];
if (n==3) varloc=memosb[3];
return varloc ;}

tpchar  __stdcall stdtitre(void)
{return "Appareil fictif\0";}

tpchar  __stdcall stddetail(void)
{return "Appareil fictif, DLL programmée en Borland C++ par P. Dieumegard, le 19 nov 2002\0";
}
```

## 4.3 Langages Basic

### 4.3.1 PureBasic et les DLL

Ce langage Basic compilé est très agréable à utiliser, et sa version commerciale permet de faire les DLL. Pour tous renseignements, consultez le site www.purebasic.fr (ou .com, ou .de).
Les versions anciennes utilisaient des nombres réels de type "simple", mais heureusement, les versions récentes (version 4 et ultérieures) permettent d'utiliser des nombres réels de type "double", ce qui permet la compatibilité avec les autres logiciels.
Voici une bibliothèque dynamique pour un appareil fictif, réalisée en PureBasic 4.2.

```
;essai de dll en Purebasic

Global nea$
Global nead$
Global nsa$
Global nsad$
Global neb$
Global nsb$

Global memosa .w
Global memosad .d
Global Dim memosb .w (8)

Global titre$

Global detail$
Global wretour .w
Global dretour .d

ProcedureDLL .l ea(n .l)
valretour=n*3
ProcedureReturn(valretour)
EndProcedure

ProcedureDLL .s nea (n .l)
If n=0
    nea$="Entrée fictive 0"
    ElseIf n=1
      nea$="Entrée fictive1"
      Else
      nea$=""
    EndIf

ProcedureReturn(nea$)
EndProcedure

ProcedureDLL .d ead(n .l)
dretour=n*2.2
ProcedureReturn(dretour)
EndProcedure

ProcedureDLL .s nead(n .l)
If n=0
  nead$="entrée (fictive) 0"
  ElseIf n=1
    nead$="entrée (fictive) 1"
    Else
```

```
  nead$=""
  EndIf
ProcedureReturn(nead$)
EndProcedure

ProcedureDLL .l sa(n .l , valeur .l)
memosa=valeur
ProcedureReturn (valeur)
EndProcedure

ProcedureDLL .s nsa (n .l)
If n=0
  nsa$="sortie analogique fictive"
  Else
  nsa$=""
  EndIf
  ProcedureReturn(nsa$)
EndProcedure

ProcedureDLL .l rsa(n .l)
ProcedureReturn(memosa)
EndProcedure


ProcedureDLL .d sad (n .l , valeur .d)
memosad=valeur
ProcedureReturn(valeur)
EndProcedure

ProcedureDLL .s nsad (n .l)
If n=0
   nsad$="sortie analogique (fictive)"
   Else
   nsad$=""
   EndIf
   ProcedureReturn(nsad$)
EndProcedure

ProcedureDLL .d rsad(n .l)
ProcedureReturn(memosad)
EndProcedure

ProcedureDLL .l eb (n .l)
If n=0
   ProcedureReturn(1)
   Else
   ProcedureReturn(0)
EndIf
EndProcedure

ProcedureDLL .s neb (n .l)
If n=0
    neb$="entrée bin. 0"
    ElseIf n=1
      neb$="entrée bin. 1"
    Else
    neb$=""
    EndIf
   ProcedureReturn neb$
EndProcedure

ProcedureDLL .l sb(n .l , valeur .l)
   memosb(n)=valeur
   ProcedureReturn(memosb)
EndProcedure

ProcedureDLL .s nsb(n .l)
If n=0
   nsb$="sortie binaire 0"
   ElseIf n=1
```

```
     nsb$="sortie binaire 1"
     ElseIf n=2
       nsb$="sortie binaire 2"
       Else
   nsb$=""
   EndIf
 ProcedureReturn(nsb$)
EndProcedure

ProcedureDLL .l rsb(n .l)
ProcedureReturn(memosb(n))
EndProcedure



ProcedureDLL .s titre()
titre$="Appareil fictif"
ProcedureReturn(titre$)
EndProcedure


ProcedureDLL.s detail()
detail$="detail de la dll fictive"
ProcedureReturn(detail$)
EndProcedure

ProcedureDLL .l stdea(n .l)
ProcedureReturn(ea(n))
EndProcedure

ProcedureDLL .s stdnea(n .l)
ProcedureReturn (nea(n))
EndProcedure

ProcedureDLL .d stdead(n .l)
;ProcedureReturn(n*1.1)
ProcedureReturn(ead(n))
EndProcedure

ProcedureDLL .s stdnead(n .l)
ProcedureReturn(nead(n))
 EndProcedure


ProcedureDLL .l stdsa(n .l , valeur .l)
ProcedureReturn(sa(n,valeur))
EndProcedure

ProcedureDLL .s stdnsa(n .l)
ProcedureReturn(nsa(n))
EndProcedure

ProcedureDLL .l stdrsa(n .l)
ProcedureReturn (rsa(n))
EndProcedure

ProcedureDLL .d stdsad(n .l , valeur .d)
ProcedureReturn(sad(n,valeur))
EndProcedure

 ProcedureDLL .s stdnsad(n .l)
 ProcedureReturn(nsad(n))
 EndProcedure

 ProcedureDLL .d stdrsad(n .l)
ProcedureReturn(rsad(n))
EndProcedure

ProcedureDLL .l stdeb(n .l)
ProcedureReturn(eb(n))
```

```
EndProcedure

ProcedureDLL .s stdneb(n .l)
ProcedureReturn (neb(n))
EndProcedure

ProcedureDLL .l stdsb(n .l , etat .l)
ProcedureReturn(sb(n,etat))
EndProcedure

ProcedureDLL .s stdnsb(n .l)
ProcedureReturn(nsb(n))
EndProcedure

ProcedureDLL .l stdrsb(n .l)
ProcedureReturn(rsb(n))
EndProcedure

 ProcedureDLL .s stdtitre()
 titre$="titre dll"
 ProcedureReturn(titre$)
 EndProcedure

 ProcedureDLL .s stddetail()
 detail$="détail de la dll fictive"
 ProcedureReturn (detail$)
 EndProcedure

 ProcedureDLL .d eadouble(x .d)
 ProcedureReturn(ead(Round(x, #PB_Round_Nearest)))
 EndProcedure

 ProcedureDLL .s neadouble(x .d)
 ProcedureReturn(nead(Round(x, #PB_Round_Nearest)))
 EndProcedure

 ProcedureDLL .d sadouble(n .d , valeur .d)
 ProcedureReturn(sad(Round(n, #PB_Round_Nearest),valeur))
 EndProcedure

 ProcedureDLL .s nsadouble(n .d)
 ProcedureReturn nsad(Round(n, #PB_Round_Nearest))
 EndProcedure

 ProcedureDLL .d rsadouble(n .d)
 ProcedureReturn rsad(Round(n, #PB_Round_Nearest))
 EndProcedure

 ProcedureDLL .d ebdouble(n .d)
 ProcedureReturn eb(Round(n, #PB_Round_Nearest))
 EndProcedure

 ProcedureDLL .s nebdouble(n .d)
 ProcedureReturn neb(Round(n, #PB_Round_Nearest))
 EndProcedure

 ProcedureDLL .d sbdouble(n .d , valeur .d)
 ProcedureReturn sb(Round(n, #PB_Round_Nearest),Round(valeur, #PB_Round_Nearest))
 EndProcedure

 ProcedureDLL .s nsbdouble(n .d)
 ProcedureReturn nsb(Round(n, #PB_Round_Nearest))
 EndProcedure

 ProcedureDLL .d rsbdouble(n .d)
 ProcedureReturn rsb(Round(n, #PB_Round_Nearest))
 EndProcedure

 ProcedureDLL .d stdeadouble(n .d)
 ProcedureReturn ead(Round(n, #PB_Round_Nearest))
```

```
 EndProcedure

 ProcedureDLL .s stdneadouble(n .d)
 ProcedureReturn neadouble(n)
 EndProcedure

 ProcedureDLL .d stdsadouble(n .d , valeur .d)
 ProcedureReturn sadouble(n,valeur)
 EndProcedure

 ProcedureDLL .s stdnsadouble(n .d)
 ProcedureReturn nsadouble(n)
 EndProcedure

 ProcedureDLL .d stdrsadouble(n .d)
 ProcedureReturn rsadouble(n)
 EndProcedure

  ProcedureDLL .d stdebdouble(n .d)
  ProcedureReturn ebdouble(n)
  EndProcedure

  ProcedureDLL .s stdnebdouble(n .d)
  ProcedureReturn nebdouble(n)
  EndProcedure

  ProcedureDLL .d stdsbdouble(n .d, valeur .d)
  ProcedureReturn sbdouble(n,valeur)
  EndProcedure

  ProcedureDLL .s stdnsbdouble(n .d)
  ProcedureReturn nsbdouble(n)
  EndProcedure

  ProcedureDLL .d stdrsbdouble(n .d)
  ProcedureReturn rsbdouble(n)
  EndProcedure


 ;initialisation éventuelle de l'appareil ?
```

### 4.3.2 FreeBasic

Normalement, il peut réaliser des DLL et les utiliser, mais la compatibilité est difficile avec les exécutables produits par d'autres langages. Assez curieusement, (version 0.4.6 pour Windows) il faut déclarer "pascal" des fonctions qui devraient être de type "stdcall".

```
public function ea pascal alias "ea"(byval n as short)as short export
    if n= 0 then
          function=0
      elseif n=1 then
            function=11
        else
              function=-777
        endif
end function

public function nea pascal alias "nea" (byval n as short) as zstring  pointer export
    if n=0 then
       function=@"entrée analogique 0"
    elseif n=1 then
           function=@" entrée analogique 1"
       else
           function=@""
       endif
end function

public function ead pascal alias "ead"  (byval n as short) as double export
    if n=0 then
```

```
          function=0
      elseif n=1 then
            function=11.11
        else
            function=-777
            endif
end function

public function nead pascal alias "nead" (byval n as short) as zstring pointer export
    if n=0 then
       function=@"entrée an. 0"
       elseif n=1 then function=@"entrée an. 1"
       else function=@""
       endif
end function

public function titre pascal alias "titre"  ()  as zstring pointer export
    function = @"titre (fictif)"
end function

public function detail pascal alias "detail"  ()  as zstring pointer export
    function = @"détail de l'appareillage fictif "
end function

public function stdea pascal alias "stdea" (byval n as short) as short export
    function = ea(n)
end function

public function stdnea pascal alias "stdnea"  (byval n as short) as zstring pointer export
    function = nea(n)
end function

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

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

public function stddetail pascal alias "stddetail" ()as zstring pointer export
    function=detail()
end function

public function stdtitre pascal alias "stdtitre" () as zstring pointer export
    function=titre()
end function
```

# 5 Utilisation de ces DLL de mesure par des logiciels d'application

## *5.1 Langages de type Pascal*

### 5.1.1 Delphi

Les essais de programmation ont été faits en Delphi 5.
Il y a deux façons de lier les fonctions des DLL au programme principal :
- la liaison statique, c'est à dire que le programme est prévu pour fonctionner avec une DLL connue dès la compilation. Ce n'est pas très intéressant ici, puisque nous souhaitons faire fonctionner le même logiciel principal avec différentes DLL, caractéristiques chacune d'un appareil de mesure.
- la liaison dynamique, où l'on peut choisir la DLL à utiliser au moment de l'exécution (à condition, bien sûr, que cette DLL contienne les bonnes fonctions !). C'est cette méthode qui sera détaillée ici.

On commence par déclarer comme variables des fonctions à charger dans la DLL, avec le bon type :

```
var l:thandle; (*type défini dans l'unité wintypes*)
var repchar : array[0..80] of char;
var stdneap : function(n:longint):pchar;stdcall;
    stdneadp :function(n:longint):pchar;stdcall;
var stdeap : function(n:longint):longint;stdcall
    stdeadp:function(n:longint):double;stdcall;
```

```
    stdcalibration(ch:pchar):pchar;stdcall;
```

puis, dans la section exécutable, on donne un contenu à ces fonctions :

```
  @stdeadp:=nil;
  @stdneadp:=nil;
  @stdcalibration:=nil;
  strpcopy(repchar,filename);
  L:=loadlibrary(repchar);
  @stdneadp:=getprocaddress(L,'stdnead');
  @stdeadp:=getprocaddress(L,'stdead');
  @stdcalibration:=getprocaddress(L,'stdcalibration');
```

éventuellement, on peut faire des tests pour savoir si la DLL est utilisable :

```
    if getprocaddress(L,'stdcalibration')<>nil
                  then reglages.enabled:=true
                  else reglages.enabled:=false;
               if (@stdeadp<>nil)and(@stdneadp<>nil) then okpiloteP:=true
                                                     else okpiloteP:=false;
```

Ensuite, les fonctions ainsi chargées pourront être utilisées normalement...
En principe, à la fin de l'utilisation de la librairie, par exemple lorsqu'on veut en charger une autre à sa place, il faut libérer la place par l'instruction FreeLibrary(L);

### 5.1.2 FreePascal

Le principe est exactement le même :

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

uses windows,sysutils,crt;

var l:thandle;
var repchar : array[0..80] of char;
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;
 @stdeadp:=nil;
  @stdneadp:=nil;
  @stdcalibration:=nil;
  strpcopy(repchar,'c:\projd5\pourfp\xadesfp.dll');
  L:=loadlibrary(repchar);
  @stdneadp:=getprocaddress(L,'stdnead');
 @stdeadp:=getprocaddress(L,'stdead');
  @stdcalibration:=getprocaddress(L,'stdcalibration');
  writeln(stdneadp(0));
 writeln(stdeadp(0));
 readln;
 end.
```

## 5.2 Langages de type C++

### 5.1.2 C++Builder (Borland)

La technique est assez semblable, mais il faut tenir compte du fait que ce logiciel semble avoir des exigences différentes de Delphi pour la déclaration des fonctions dans la DLL.
Comme pour Delphi, on commence par déclarer les types, variables et fonctions à utiliser, en se souvenant que la casse des caractères a de l'importance (ne pas confondre minuscules et majuscules) :

```
   typedef char  *(*stddetailtype) ();
   typedef char *(*stdtitretype) ();
   typedef double (*stdeadtype) (unsigned short numvoie);
   typedef char *(*stdneadtype) (unsigned short numvoie);
```

```
   stddetailtype stddetail;
   stdtitretype stdtitre;
   stdeadtype stdstdead;
   stdneadtype stdstdnead;
int a;
HINSTANCE handle=NULL;
char  repchar[60];
String machaine;
```
puis on charge la librairie :
```
StrPCopy(repchar,"c:\\projd5\\xadestar.dll");
handle=LoadLibrary(repchar);
stddetail = (stddetailtype)GetProcAddress(handle,"stddetail");
stdtitre = (stdtitretype)GetProcAddress(handle,"stdtitre");
stdead = (stdeadtype)GetProcAddress(handle,"stdead");
stdnead = (stdneadtype)GetProcAddress(handle,"stdnead");
//Caption=stddetail();         //correct
Caption=stdtitre();//correct
//Caption=stdnead(0);     //correct;
//Caption=String(stdead(0));  //correct;
```

et à la fin de l'utilisation, on peut libérer la mémoire :
```
FreeLibrary(handle);
```

## *5.3 Langages de type Basic*

### 5.3.1 PureBasic

La version de démonstration (gratuite) est un peu bridée par rapport à la version complète (payante). Dans cette version de démonstration, on ne peut pas compiler de DLL ni faire de programmes exécutables autonomes, mais on peut utiliser les DLL déjà faites, et exécuter un programme PureBasic à partir de l'environnement de développement.

On peut ajouter quelques points très agréables :
- PureBasic a les menus et l'aide en français (ou en anglais, ou en allemand...)
- La syntaxe est automatiquement vérifiée et mise en évidence lors de la frappe

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, on l'appelle par CallFunction.
Pour les fonctions avec paramètres, il faut utiliser celles qui sont compilées avec l'option stdcall (donc celles dont le nom commence par std dans la nomenclature décrite précédemment).
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, on ferme la bibliothèque par CloseLibrary.

```
 ; Voici le programme client qui utilise la DLL
  ;
  nomdll$="dll_de~1.dll"


Global string .s
Procedure .s detail()
   string =PeekS(CallFunction(0,"detail"))
   ProcedureReturn (string)
   EndProcedure

 Procedure .l ea (n .l)
 ProcedureReturn(CallFunction(0,"stdea",n))
EndProcedure
```

```
Procedure .s nea(n .l)
string=PeekS(CallFunction(0,"stdnea",n))
ProcedureReturn(string)
EndProcedure

 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$)
 numfonc=CountLibraryFunctions(0)
 Print(Str(numfonc))
 ExamineLibraryFunctions(0)
 For i=1 To numfonc
    NextLibraryFunction()
    PrintN (LibraryFunctionName())
    Next i
 PrintN(nea(1) +"   "+Str(ea(1)))
 Print(" et ead")
 Print(nead(1)+"    "+StrD(ead(1),2))
PrintN("")
Print (detail())

 CloseLibrary(0)
 Else
 Print("problème !")
 EndIf
  Input()
CloseConsole()
```

Les premières versions de PureBasic ne permettaient pas de travailler avec les réels de type "double", mais les versions récentes (PureBasic 4 et suivantes) le permettent, ce qui assure une bonne compatibilité avec les autres langages de programmation.

### 5.3.2 FreeBasic

On peut utiliser des DLL faites par Delphi ou Purebasic par des programmes écrits en FreeBasic,

```
      dim library as integer
        dim ea as function (byval n as short) as short
    dim ead as function (byval n as short) as double
    dim nea as function (byval n as short) as zstring pointer
    dim nead as function (byval n as short) as zstring pointer
    dim detail as function () as zstring pointer
    dim stdea as function (byval n as short) as short
    dim stdnea as function (byval n as short) as zstring pointer
    dim stdead as function (byval n as short) as double
    dim stdnead as function(byval n as short) as zstring pointer
    dim stddetail as function () as zstring pointer

      library = dylibload( "dll_pb2" )rem pour charger la bibliothèque dll_pb2.dll
      if( library = 0 ) then
            print "bibliothèque inchargeable !"

      end if
      ea=dylibsymbol(library,"ea")
    if (ea=0) then
```

```
        print "la fonction ea n'est pas utilisable"
        end if
    nea=dylibsymbol(library,"nea")
    ead=dylibsymbol(library,"ead")
    nead=dylibsymbol(library,"nead")
    detail=dylibsymbol(library,"detail")
    stdea = dylibsymbol(library,"stdea")
    stdnea=dylibsymbol(library,"stdnea")
      stdead = dylibsymbol(library,"stdead")
    stdnead=dylibsymbol(library,"stdnead")
    stddetail=dylibsymbol(library,"stddetail")

    print "sans std"
    print *detail()
    print *nea(1);ea(1)
    print *nead(1);ead(1)
    print
    print "avec std"
    print *stddetail()
    print *stdnea(1);stdea(1)
    print *stdnead(1);stdead(1)
    input "",z

            dylibfree library
```

## 5.4 MSW-Logo

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 pour les habitués de Delphi :
- 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.

## 5.5 Python

Python est un langage interprété, existant pour Windows comme pour Linux (et d'autres systèmes d'exploitation).
Si on suppose que la bibliothèque dynamique est dans le répertoire de travail, on peut l'appeler assez facilement.

```
from ctypes import *
Windll.LoadLibrary(« xmadll.dll »)
fdetail=windll.xmadll.stddetail
fdetail.restype=c_char_p
print fdetail()
```

## 5.6 Scripts de logiciels de bureautique

### 5.6.1 OpenBasic, pour la suite bureautique OpenOffice (et StarOffice)

La suite bureautique OpenOffice est disponible en plusieurs langues, puissante, et dispose du langage de programmation OpenBasic. Tous ces avantages compensent le fait que ce soit un logiciel assez lent

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.

Note pour les habitués de Delphi : comme pour d'autres logiciels, il faut tenir compte de la casse des noms des variables,
Si les fonctions sont compilées avec stdcall, l'ordre des paramètres est le même que celui dans la déclaration dans Delphi ; par contre, si les fonctions ne sont pas compilées avec stdcall, l'ordre est inversé.

### 5.6.2 Lotus Smartsuite 97

Le langage LotusScript est un Basic assez proche de StarBasic.
Il veut que les paramètres soient déclarés byval
Il veut les paramètres de type double, ou bien déclaration en std;

- std : ordre normal
- double non std : ordre inverse
- std et double : ordre normal

```
Declare Function neadouble Lib "C:\projd5\mgw32\pilotes\xadestar.dll" _
Alias "neadouble" (Byval n As Double) As String

Declare Function eadouble Lib "c:\projd5\mgw32\pilotes\xadestar.dll" _
Alias "eadouble" (Byval n As Double) As Double

Declare Function stdsbdouble Lib "c:\projd5\mgw32\pilotes\xadestar.dll" _
```

```
Alias "stdsbdouble" (Byval voie As Double , Byval valeur As Double ) As Double

Declare Function stdsb Lib "c:\projd5\mgw32\pilotes\xadestar.dll" _
Alias "stdsb" (Byval voie As Integer , Byval valeur As Integer) As Integer

Declare Function sbdouble Lib "c:\projd5\mgw32\pilotes\xadestar.dll" _
Alias "sbdouble" (Byval voie As Double, Byval valeur As Double) As Double

Declare Function sb Lib "c:\projd5\mgw32\pilotes\xadestar.dll" _
Alias "sb" (Byval voie As Integer, Byval valeur As Integer) As Integer

Sub Click(Source As Buttoncontrol)
      Messagebox Str$(stdsb(1,1)),MB_OK,"Demo"

End Sub
```

### 5.6.3 Tableur Microsoft-Excel, et langage VisualBasic

Le principe de la programmation est assez voisin de StarBasic.

```
Sub Module1
Declare Function stdead Lib "d:\dieumeg\xadestar.dll" (ByVal n As Integer) As Double
Declare Function stdnead Lib "d:\dieumeg\xadestar.dll" (ByVal n As Integer) As String

Sub MacroD()
Rem ' MacroD Macro
Rem ' Macro enregistrée le 31/01/03 par biologie
Rem '
Rem ' Touche de raccourci du clavier: Ctrl+Maj+D

Rem     'ActiveCell.FormulaR1C1 = "=RC[2]"
Rem     'Range("A2").Select
Rem     'ActiveCell.Value = stdnead(1)
Rem     ActiveCell.Value = 55
Rem End Sub
Rem '
Rem ' MacroE Macro
Rem ' Macro enregistrée le 31/01/03 par biologie
Rem '
Rem ' Touche de raccourci du clavier: Ctrl+Maj+E
Rem '
Rem     ActiveCell.Offset(-5, 4).Range("A1").Select
Rem     ActiveCell.Value = 44
Rem     ActiveCell.Value = stdnead(0)
Rem End Sub
Rem
End Sub
```

## 5.7 Logiciels de calcul numérique

### 5.7.1 Freemat

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).

La version 3.6 de Freemat accepte les appels de paramètres de type cdecl, pascal, register et stdcall, mais la version 4 n'accepte que les fonctions de type cdecl.

# 6 Que faire pour les logiciels qui ne peuvent pas utiliser les librairies à liaison dynamique ?

Certains logiciels ne semblent pas prévus pour utiliser des fonctions de librairies extérieures. C'est le cas de certains langages de programmation simples tels que SmallBasic, qui est un langage de type Basic pouvant fonctionner sous divers systèmes d'exploitation (Windows, Linux, PalmOS...)
D'autres peuvent théoriquement le faire, mais la façon de le faire est incompréhensible, ce qui revient au même. C'est le cas en particulier des logiciels mathématiques comme Scilab et Matlab, qui sont très puissants pour traiter les séries de valeurs numériques. C'est le cas aussi de Python, langage de programmation à tout faire.
Heureusement, ces logiciels sont souvent capables de lancer des programmes extérieurs.

On peut donc imaginer de faire un petit programme (Delphi, C++ ou équivalent) capable d'appeler les DLL et de faire les mesures, puis d'envoyer ces mesures, directement ou indirectement, vers le logiciel principal. Consultez le texte sur les pilotes exécutables.

Tous les langages de programmation, et normalement tous les logiciels ayant des fonctions de programmation, permettent de lancer un programme extérieur.
Pour certains de ces logiciels, le programme extérieur fait simplement son travail, puis s'arrête. Aucune information, ou très peu, n'est renvoyée vers le logiciel appelant. Dans ce cas, ce n'est pas très simple de faire un pilote. Il faut que le pilote fasse la mesure, puis qu'il enregistre le résultat dans un fichier, et qu'ensuite le logiciel principal lise le résultat dans le fichier.
Heureusement, il existe des logiciels qui récupèrent directement dans un tableau de chaînes de caractères tous les caractères qui normalement auraient dû être envoyés vers l'écran par le pilote.
Par exemple, supposons que le pilote doit écrire la chaîne de caractère «5.555» à l'écran, par l'instruction «writeln» de Pascal, ou «print» de Basic, ou «printf» de C++. Lorsque ce pilote est appelé par le logiciel, celui-ci récupère directement cette chaîne de caractère dans une variable. Il suffira donc ensuite de traiter cette chaîne de caractères par le logiciel, en la transformant en valeur numérique... C'est très pratique.

### 6.1.1 Un programme de pilote exécutable en Delphi, appelant une bibliothèque dynamique, et envoyant directement les valeurs vers l'écran

```
program pdll32;
(*programme appelant une bibliothèque dynamique de mesure, et envoyant le résultat
à l'écran : à utiliser avec les logiciels interceptant les messages pour l'écran*)
(* conçu initialement pour Scilab*)
{$APPTYPE CONSOLE}
uses
  sysutils,wintypes;

  var l:thandle;
  var repchar:array[0..80] of char;
  var nead:function(n:word):pchar;
  var ead:function(n:word):double;
  var stdead:function(n:word):double;    stdcall;
  var stdnead:function(n:integer):pchar;    stdcall;
  var detail:function : pchar;
  var chemindll,nomfonction,chparam1,chparam2:string;
  var valparam1:double;
  var valparam2:double;
  var i:integer;

 begin
   chemindll:=paramstr(1);
  nomfonction:=paramstr(2);
  if paramcount>=3 then valparam1:=strtofloat(paramstr(3));
  if paramcount>=4 then valparam2:=strtofloat(paramstr(4));
  strpcopy(repchar,chemindll);
  @nead:=nil; @ead:=nil;@stdead:=nil;@stdnead:=nil;@detail:=nil;
   l:=loadlibrary(repchar);
   @nead:=getprocaddress(L,'nead');
   @ead:=getprocaddress(L,'ead');
   @stdead:=getprocaddress(L,'stdead');
   @stdnead:=getprocaddress(L,'stdnead');
   @detail:=getprocaddress(L,'detail');
   if nomfonction='ead' then writeln(ead(round(valparam1)):10:5);
   if nomfonction='nead' then writeln(nead(round(valparam1)));
```

```
   if nomfonction='stdead' then writeln(stdead(round(valparam1)):10:5);
   if nomfonction='stdnead' then writeln(stdnead(round(valparam1)));
   if nomfonction='detail' then writeln(detail);
  freelibrary(L);
end.
```

# 7 Et pour Linux ?

Les DLL pour Windows sont peu différentes des objets partagés de Linux. Après quelques modifications, elles peuvent être compilées par Kylix. (lorsqu'elles sont écrites en Pascal) ou g++ (lorsqu'elles sont écrites en C++).

## *7.1 le problème d'utilisation des ports en Linux*

Linux est un système d'exploitation conçu pour être multi-tâches et multi-utilisateurs. Normalement, il y a un super-utilisateur (administrateur du réseau), nommé «root», qui peut commander complètement l'ordinateur, et les autres utilisateurs, qui n'ont que des droits réduits.
Les utilisateurs normaux n'ont pas le droit d'accéder directement aux ports de l'ordinateur, ce qui est dans l'absolu une bonne mesure de sécurité, car l'accès direct à l'électronique permet de faire beaucoup de bêtises. Malheureusement, pour beaucoup d'appareils de mesure, en particulier pour les cartes insérées dans les connecteurs d'extension, il est indispensable de pouvoir écrire et lire des valeurs sur les ports.
Une solution est bien sûr d'utiliser l'ordinateur en tant que super-utilisateur «root». Mais dans ce cas, il n'y a aucune protection contre les erreurs !

Le problème est donc d'autoriser l'utilisateur normal à accéder à certains ports, ceux utiles pour la mesure.

La solution que je propose est la suivante :
- écrire et compiler les bibliothèques dynamiques en tant que «root», qui sera donc le légitime propriétaire et utilisateur de cette bibliothèque dynamique.
- ensuite, toujours en tant que «root», modifier les droits d'accès de cette bibliothèque dynamique, en déclarant que tous les utilisateurs pourront s'en servir, avec les mêmes droits que «root».

Deux méthodes sont possibles pour donner cette autorisation :
- la méthode «ligne de commande» (Merci à Georges Khaznadar): si la bibliothèque à utiliser s'appelle liblxmachine.so, il faut entrer la commande :
chmod 4755 liblxmachine.so
Le nombre mystérieux 4755 signifie que toute personne ayant droit d'exécution a les mêmes droits que le propriétaire initial, et que tout utilisateur aura le droit d'exécution.

- la méthode «cliquage de souris» avec les versions modernes de Linux, comme par exemple Mandrake 8.
En mode graphique, lorsqu'on clique avec le bouton droit de la souris sur l'icône représentant le fichier liblxmachine.so, on peut modifier les propriétés de ce fichier, et donc donner les droits précédents.

Ensuite, après ces opérations faites en tant que «root», les utilisateurs normaux pourront utiliser la bibliothèque liblxmachine.so qui accèdera aux ports de mesure.

## *7.2 Ecriture d'objets partagés*

### 7.2.1 Langages Pascal

#### *7.2.1.1 Kylix*

Le fichier ainsi compilé peut être appelé par un logiciel d'application écrit en Kylix, avec à peu près la même syntaxe qu'avec Delphi pour Windows.
Le fichier ci-dessous correspond à une librairie simple, facilement utilisable en Kylix. Pour pouvoir l'utiliser avec d'autres logiciels et langages de programmation, il est probable qu'il faut ajouter d'autres fonction (stdead, stdnead, etc), comme pour les langages de programmation de Windows.

```
library xadestar_linux;

uses
```

```
  SysUtils,uades1,
  Classes;

const NumFuncts = 17;
type array100= array[0..100] of char;
type pdouble=^double;

var memosb:array[0..3]of boolean;
    memosad:double;
function ea(n:longint):longint;
var o:integer;
n1,n2:word;
begin
  o:=n;
  ea:=-777;
  if (n=0)or(n=1)then lit_niveau(n1,n2);
  case o of 0: ea:=n1;
            1: ea:=n2;
            2: ades_off;
            3: ades_on;
            end;
end;


function nea(n:longint):pchar;
var chloc:string[20];  tabcarloc:array[0..20] of char;
begin
  str(n,chloc);
  if (n<2) and (n>-1)
    then strpcopy(tabcarloc,'CAN voie '+chloc)
    else if n=2 then strpcopy(tabcarloc,'RAZ pour impression')
       else if n=3 then strpcopy(tabcarloc,'Réinitialisation')
          else strpcopy(tabcarloc,'');
  nea:=tabcarloc;
end;


function ead(n:longint):double;
var aux:integer;

begin
  if (n=0) or (n=1)or (n=2) or (n=3)
     then begin
       aux:=ea(n);
       ead:=(5.0*aux)/1024;
          end
     else  ead:=-777;
end;

function nead(n:longint):pchar;
var chloc:string[20];  tabcarloc:array[0..20] of char;
begin
  if (n=0)
    then nead:='E. analogique 0'
      else if n=1 then nead:='E. analogique 1'
      else if n=2 then nead:='RAZ pour impression'
             else if n=3 then nead:='Réinitialisation'
    else nead:='';
end;


function sa(n:longint; valeur:longint):longint;
begin
 sa:=-777;
end;
function nsa(n:longint):pchar;
  var tabcarloc:array[0..20] of char;
begin
if n=0 then
  nsa:='SA fictive'+chr(0)
```

```
  else nsa:=''+chr(0);
end;


function sad(n: longint ; valeur:double):double;
begin
MEMOsad:=valeur;
sad:=0;
end;

function nsad(n:longint):pchar;
  var tabcarloc:array[0..20] of char;
begin
   if n=0 then nsad:='SA fictive'+chr(0)
      else nsad:=''+chr(0);
end;

function rsa(n:longint):longint;
begin
if n=0 then rsa:=2 else rsa:=-777;
end;

 function rsad(n:longint):double;
begin
  if n=0 then rsad:=2 else rsad:=-777;
end;


function eb(n:longint):longint;
begin
 if n=0 then eb:=0
        else if n=1 then eb:=1
                    else eb:=-777;
end;

function neb(n:longint):pchar;
var  tabcarloc:array[0..20] of char;
begin
 if n=0 then neb:='EB fictive 0'+chr(0)
        else if n=1 then neb:='EB fictive 1'+chr(0)
                    else neb:=chr(0);
end;

Const tabmasque : array[0..7] of byte = (1,2,4,8,16,32,64,128);
   memosl1:byte=0;
 function testebit(nombre:word ; numbit:byte):boolean;
begin    testeBit:=odd(nombre shr numbit);end;

procedure forcebit(var octet :byte ; numbit:byte ; etat:boolean);
begin
  if etat then octet :=octet or tabmasque[numbit]
          else octet :=octet and (not tabmasque[numbit]);
end;

function sb(n:longint; etat:longint):longint;
begin
 case n of
   0..3 : begin relais[n+1]:=(etat=1);
   sb:=n*100+etat;
   pilote_relais(0);
         //forcebit(memosl1,n,(etat=1));
   memosb[n]:=(etat=1);
          end;
   else sb:=-777;
    end;
end;



function nsb(n:longint):pchar;
```

```
var chloc:string[20];  tabcarloc:array[0..20] of char;
begin
result:=chr(0);
if n=0 then result:='relais A'+chr(0);
if n=1 then result:='relais B'+chr(0);
if n=2 then result:='relais C'+ chr(0);
if n=3 then result:='relais D'+chr(0);
end;

function rsb(n:longint):longint;
begin
 (* if n in [0..3]
     then if testebit(memosl1,n)
           then rsb:=1
           else rsb:=0
     else rsb:=-777 ;
     *)
 if n in [0..3] then if memosb[n] then rsb:=1 else rsb:=0
                else rsb:=-777;
end;


function titre : pchar;
begin titre:='ADES en LPT1:';end;
function detail : pchar;
begin detail:='DLL (32 bits) pour ADES 10 bits en LPT1:, par P. Dieumegard, le 20
février 2001';end;

(*les deux fonctions qui suivent sont pour EXCEL*)
function peaD(n:longint):pdouble;
var dloc:double;longint
begin dloc:=eaD(n);pead:=@dloc;end;

function psad(n:longint;valeur:pdouble):pdouble;
var dloc1,dloc2:double;
begin dloc1:=valeur^;dloc2:=SAD(n,dloc1);psad:=@dloc2;end;


(*les fonctions " double " qui suivent sont pour StarOffice*)
function eadouble(x:double):double;
var  nloc:word;
begin
  nloc:=round(x);
  eadouble:=ead(nloc);
end;

function neadouble(x:double):pchar;
var nloc:word;
begin
  nloc:=round(x);
  neadouble:=nead(nloc);
end;

function sadouble(x:double;xval:double):double;
begin sadouble:=sad(round(x),xval);end;

function nsadouble(x:double):pchar;
begin nsadouble:=nsad(round(x));end;

function rsadouble(x:double):double;
begin rsadouble:=rsad(round(x));end;

function ebdouble(x:double):double;
begin ebdouble:=eb(round(x));end;

function nebdouble(x:double):pchar;
begin nebdouble:=neb(round(x));end;

function sbdouble(n:double;etat:double):double;
begin sbdouble:=sb(round(n),round(etat));end;
```

```
function nsbdouble(n:double):pchar;
begin nsbdouble:=nsb(round(n));end;

function rsbdouble(n:double):double;
begin rsbdouble:=rsb(round(n));end;


function stdea(n:longint):longint;  begin stdea:=ea(n);end;stdcall;
function stdnea(n:longint):pchar;  begin stdnea:=nea(n);end;stdcall;
function stdead(n:longint):double;  begin stdead:=ead(n);end;stdcall;
function stdnead(n:longint):pchar; begin stdnead:=nead(n);end;stdcall;
function stdsa(n: longint ;valeur:longint):longint;
      begin stdsa:=sa(n,valeur);end;stdcall;
function stdnsa(n:longint):pchar; begin stdnsa:=nsa(n);end;stdcall;
function stdrsa(n:longint):longint; begin stdrsa:=stdrsa(n);end;stdcall;
function stdsad(n:longint; valeur:double):double; begin
      stdsad:=sad(n,valeur);end;stdcall;
function stdrsad(n:longint):double; begin stdrsad:=rsad(n);end;stdcall;
function stdeb(n:longint):longint; begin stdeb:=eb(n);end;stdcall;
function stdneb(n:longint):pchar; begin stdneb:=neb(n);end;stdcall;
function stdsb(n:longint;etat:longint):longint; begin stdsb:=sb(n,etat);end;stdcall;
function stdnsb(n:longint):pchar; begin stdnsb:=stdnsb(n);end;stdcall;
function stdrsb(n:longint):longint; begin stdrsb:=stdrsb(n);end;stdcall;
function stdtitre:pchar; begin stdtitre:=titre;end;stdcall;
function stddetail:pchar; begin stddetail:=detail;end;stdcall;
(*stdtitre et stddetail ne sont pas indispensables, car ils  sont sans paramètres*)
(* comme titre et detail*)
exports
  ea ,  nea ,ead ,nead,
  sa ,nsa,sad ,  nsad ,rsa,rsad,
  eb ,  neb ,  sb ,  nsb ,rsb,
  titre , detail,
  pead ,  psad ,
  stdea,stdnea,stdead,stdnead,
  stdsa,stdnsa,stdrsa,stdsad,stdnsad,stdrsad,
  stdeb,stdneb,stdsb,stdnsb,stdrsb,
  eadouble ,  neadouble ,
  sadouble,nsadouble,rsadouble,
  ebdouble,nebdouble,
  sbdouble, nsbdouble,rsbdouble;

var i:word;
begin
 nbitades:=10;
 initialise_ades(888);
 ades_on;
 for i:=0 to 3 do memosb[i]:=false;
end.
```

### *7.2.1.2 en FreePascal*

Bien sûr, c'est du Pascal, et la bibliothèque doit être rédigée presque comme en Delphi, mais pas de façon tout à fait identique. Voici le début et la fin d'une bibliothèque pour ADES.

```
library lxades1;
//bibliothèque dynamique pour Linux et Ades, à utiliser avec FreePascal

uses
  SysUtils,linux,
  uades1;

type array100= array[0..100] of char;
type pdouble=^double;

var memosb:array[0..3]of boolean;
    memosad:double;
function ea(n:word):integer; pascal;export;
var o:integer;
n1,n2:word;
```

```
begin
  o:=n;
  ea:=-777;
  if (n=0)or(n=1)then lit_niveau(n1,n2);
  case o of 0: ea:=n1;
            1: ea:=n2;
            2: ades_off;
            3: ades_on;
            end;
end;
```

Les fonctions de la bibliothèque sont identiques à celles de Kylix, mais à la fin, il faut ajouter une ligne pour donner le droit d'accéder aux ports :

```
var i:word;
begin
 ioperm(888,3,1);
 nbitades:=10;
 initialise_ades(888);
 ades_on;
 for i:=0 to 3 do memosb[i]:=false;
end.
```

La ligne ioperm(888,3,1) signifie «permettre l'entrée et la sortie d'information à partir de l'adresse de port 888 (qui est l'adresse de base de l'interface parallèle utilisée par ADES), sur 3 octets successifs (l'adresse de lecture et d'écriture d'information sont deux adresses successives)». Cette instruction ioperm sera à utiliser pour tous les appareils de mesure devant accéder aux ports, avec bien sûr des valeurs différentes selon les ports à utiliser.

### 7.2.2 Ecriture de bibliothèque dynamique en C++

Comme pour Windows, voici une bibliothèque pour une interface fictive. Pour l'utiliser avec un véritable appareil de mesure, il faut remplacer les valeurs arbitraires des différentes fonctions par la vraie mesure ou la vraie commande.

Cette bibliothèque a été rendue possible grâce aux renseignements fournis sur le site http://www.wachtler.de/dynamischeBibliotheken.

Si son nom est lxdllcpp.cpp, il faut la compiler avec la ligne de commande suivante :

```
g++ -shared lxdllcpp.cpp -o liblxdllcpp.so
```

```
int memosa;

double memosad;
bool memosb[4];

typedef  char * tpchar;
//pour avoir la correspondance avec le type pchar de Delphi

static int
Wert = 0;
extern "C" int summe( int a, int b )
{
  return a + b;
}

extern "C" int immereinsmehr( void )
{
  if( Wert>6 )
    {
      throw( 125 );
    }   return Wert++;
}

extern "C" int  stdea(int n)
{ if (n>-1&&n<3) {return n*3;}else {return -777;}}

extern "C" tpchar  stdnea(int n)
{
tpchar chloc;
```

```
 chloc="\0";
 if (n==0) chloc="entrée analogique 0\0";
 if (n==1) chloc="EA 1 (volts)\0";
 if (n==2) chloc="température °C\0";
   return chloc; }


extern "C" double   stdead(int n)
{
double varloc;
if (n>-1&&n<3) {varloc= n*3.33;}else{varloc= -777;}
return varloc;
 }

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

extern "C" int  stdsa(int n , int val)
{ if (n==0) {memosa=val ;return val;}else{return -777;}
}

extern "C" tpchar  stdnsa(int n)
{ tpchar chloc;
if (n==0) chloc="sortie analogique\0"; else chloc="\0";
return chloc; }


extern "C" double  stdsad(int n, double val)
{ if (n==0){memosad=val; return val;} else return -777;
}

extern "C" tpchar   stdnsad(int n)
{if (n==0) return "SA 0 (volts)\0"; else return "\0"; }

extern "C" int   stdrsa(int n)
{ if (n==0) return memosa; else return -777; }

extern "C" double  stdrsad(int n)
{ if(n==0) return memosad; else return -777;}

extern "C" int  stdeb(int n)
{
if (n==0)
        return 0;
        else if (n==1)
                return 1;
                else return -777;
}

extern "C" tpchar  stdneb(int n)
{ if (n==0)
        return "EB fictive 0\0";
        else if (n==1)
                return "EB fictive 1\0";
                else return "\0";
}

extern "C" int   stdsb(int n, int etat)
{ int varloc;
  varloc=-777;
//mettre ici la commande des sorties binaires,
// ainsi que la mémorisation dans le tableau memosb;
  if (n==0) varloc=0;
  if (n==1) varloc=1;
```

```
  if (n==2) varloc=1;
  if (n==3) varloc=0;
  return varloc;
  }

extern "C" tpchar  stdnsb(int n)
{tpchar varloc;
varloc=="\0";
if (n==0) varloc="Relais 0\0";
if (n==1) varloc="Relais 1\0";
if (n==2) varloc="Relais 2\0";
if (n==3) varloc="Relais 3\0";
return varloc;  }

extern "C" int  stdrsb(int n)
{int varloc;
varloc=-777;
if (n==0) varloc=memosb[0];
if (n==1) varloc=memosb[1];
if (n==2) varloc=memosb[2];
if (n==3) varloc=memosb[3];
return varloc ;}


extern "C" tpchar  stdtitre(void)
{return "Appareil fictif\0";}

extern "C" tpchar   stddetail(void)
{return "Appareil fictif, DLL programmée en Borland C++ par P. Dieumegard, le 19 nov
2002\0";
}
```

## 7.3 Appel de la librairie (objet partagé) par les langages de programmation

### 7.3.1 Langages Pascal

#### 7.3.1.1 Kylix

Le principe est exactement le même qu'en Delphi pour Windows : on met les instruction de chargement de la bibliothèque :

```
dialouvrepil.execute;
   //messagedlg(dialouvrepil.filename, mtconfirmation,[mbyes],0);
     if dialouvrepil.filename <>''
      then begin
      nomdll:=changefileext(dialouvrepil.filename,'');
     @detailp:=nil;@calibrationp:=nil; @sad:=nil;
     @nsad:=nil;   @titre:=nil;
     @eadp:=nil;
     @neadp:=nil;
     @sb:=nil;@nsb:=nil;  @rsad:=nil; @rsb:=nil;
     @eb:=nil;@neb:=nil;
            strpcopy(repchar,dialouvrepil.filename);
            L:=loadlibrary(repchar);
                 nompilotep:=dialouvrepil.filename;
               @detailp:=getprocaddress(L,'detail');
               @titre:=getprocaddress(L,'titre');
               @neap:=getprocaddress(L,'nea');
               @eap:=getprocaddress(L,'ea');
               @neadp:=getprocaddress(L,'nead');
               @eadp:=getprocaddress(L,'ead');
               @sad:=getprocaddress(L,'sad');
               @nsad:=getprocaddress(L,'nsad');
               @sb:=getprocaddress(L,'sb');
               @nsb:=getprocaddress(L,'nsb');
               @eb:=getprocaddress(L,'eb');
               @neb:=getprocaddress(L,'neb');
               @rsad:=getprocaddress(L,'rsad');
               @rsb:=getprocaddress(L,'rsb');
```

```
                @calibrationp:=getprocaddress(L,'calibration');
                if getprocaddress(L,'calibration')<>nil
                   then reglages.enabled:=true
                   else reglages.enabled:=false;
                if (@eadp<>nil)and(@neadp<>nil) then okpiloteP:=true
                                                else okpiloteP:=false;
if @titre<>nil
    then begin form1.caption:=strpas(titre);
         boutonfichiers.enabled:=true;
         end;
```

#### 7.3.1.2 *FreePascal*

Le principe est le même que pour Kylix, sauf que les fonctions getprocaddress n'existent pas en standard, non plus que loadlibrary ou freelibrary.
Il faut donc les créer, dans le corps du programme.

```
function dlopen(afile:pchar;mode:longint):pointer;cdecl;external 'dl';
function LoadLibrary(name:pchar):thandle;
begin
   Result:=LongWord(dlopen(name,$101 {RTLD_GLOBAL or RTLD_LAZY}));
  end;

function dlclose(handle:pointer):longint;external 'dl';
procedure FreeLibrary(handle:thandle);
begin dlclose(pointer(handle));end;

function dlsym(handle:pointer;name:pchar):pointer;external 'dl';
function getprocaddress(handle:thandle;name : pchar):pointer;
//begin result:=dlsym(pointer(handle),name);end;
begin result:=dlsym(pointer(handle),name );end;
```

### 7.3.2 Langages C et C++

Le langage C est le langage de référence pour Linux.
Voici un petit programme appelant une bibliothèque de mesure, et affichant les fonctions «detail», «stdead(1)» et «stdnead(1)».

```
//testdll_linux.c : essai de programme en C appelant les bibliothèques .so
//à compiler par
//g++ -rdynamic -ldl -o testdll_linux testdll_linux.c
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <string>

main()
{
typedef char *(*detailtype)();
typedef double (*stdeadtype) (int numvoie);
typedef char *(*stdneadtype) (int numvoie);
void *handle=NULL;
detailtype detail;
stdeadtype stdead;
stdneadtype stdnead;
double mondouble;
char repchar[60];
char * moncar;
int a;

handle=dlopen("./liblxdllvide.so",RTLD_NOW);
detail=(detailtype)dlsym(handle,"detail");
stdead=(stdeadtype)dlsym(handle,"stdead");
stdnead=(stdneadtype)dlsym(handle,"stdnead");

mondouble=stdead(2);
mach2="44.44";
moncar=detail();
printf(moncar);
```

```
printf("\n");
machaine="simple essai de chaîne";
gcvt(mondouble,5,repchar);
printf(repchar);
printf(stdnead(1));
return 0;
}
```

### 7.3.3 Langages Basic

### 7.3.4 Problèmes pour divers logiciels ou langages de programmation

Normalement, les homologues pour Linux des logiciels pour Windows pouvant utiliser les DLL devraient pouvoir utiliser les objets partagés .so.
Dans la réalité, c'est moins simple...

- StarOffice et OpenOffice existent aussi en version Linux, mais ne semblent pas capables d'appeler les fichiers .so

Comment faire ?
La seule solution pour StarOffice et PureBasic semble d'utiliser les «pilotes enregistrant leurs résultats dans un fichier», tels qu'ils sont décrits pour Windows. Ce n'est pas très simple.

Scilab : là aussi, il faut utiliser les pilotes exécutables (la documentation est très confuse pour l'emploi de bibliothèques dynamiques).

# 8 Bref résumé des compatibilités Windows/Linux pour ce qui est des bibliothèques dynamiques et des pilotes d'appareils de mesure :

- Pour écrire les bibliothèques : les langages Pascal (Delphi, Kylix, FreePascal...), Basic (PureBasic) et C++ (C++Builder, g++...) sont utilisables, aussi bien sous Windows que sous Linux. .

- Pour utiliser directement les bibliothèques :
Sous Windows et Linux, les mêmes langages Pascal , Basic et C++ sont utilisables (Delphi, Kylix, PureBasic, C++Builder, g++...)
Sous Windows, d'autres langages permettent d'utiliser les DLL : MSW-Logo, StarOffice-StarBasic, Lotus SmartSuite (Lotus Script), Python...
- Certains logiciels ne permettent pas (ou difficilement) d'utiliser les bibliothèques dynamiques, mais permettent de récupérer directement dans des chaînes de caractères (ou des tableaux de chaînes) les messages envoyés normalement à l'écran par les programmes qu'ils ont lancés. Dans ce cas, on peut utiliser un «pilote exécutable» sous la forme d'un programme exécutable qui reçoit des précisions en ligne de commande, et qui renvoie le résultat des opérations sous la forme d'une chaîne de caractères à l'écran.
Ces programmes exécutables peuvent être des programmes autonomes, mais aussi et surtout, des programmes écrits en Pascal ou C, et qui eux-mêmes appellent une bibliothèque dynamique dont le nom est passé en paramètre, avant les autres paramètres décrivant les voies à mesurer ou les valeurs à fixer. Ceci permet, indirectement, d'utiliser les bibliothèques dynamiques pour les logiciels ne pouvant pas les appeler directement.
Sous Windows et Linux, c'est le cas de Scilab, de SmallBasic et YaBasic.

(il semblerait que PHP ait aussi de telles caractéristiques, grâce à l'instruction system($commande)... à creuser...