---------- Fichier IPElectro.pas ----------

Avec les informations ci-dessous et les fichiers PDF vous pourrez concevoir une DLL pour PElectro.

Constantes PElectro

const

Les messages suivants permettent aux DLLs de communiquer avec le programme PElectro.

```
WM_LOCKFUNC= WM_USER;
```

Le message LOCKFUNC permet de verrouiller/déverrouiller une fonction suivant les paramètres LockNew, LockOpen, LockAffi, LockCCP et LockNCompo qui sont décrits plus loin. Le paramètre LParam à 1 permet de déverrouiller la fonction, LParam à 0 verrouille la fonction.
Exemple:
r1 := SendMessage(Hdle,WM_LOCKFUNC,LockAffi+LockNew,0); verrouille les fonctions LockAffi et LockNew. Si le paramètre avait été à 1, il s'agirait d'un déverrouillage.
Le retour du message est normalement 1.

```
WM_LANGUID = WM_USER+1;
```

Le message WM_LANGUID permet de définir ou de connaitre la langue utilisée. Si WParam est à 0, le retour du message est le code de la langue, par exemple le Français (définit à $0C plus loin).
Si WParam est différent de 0, sa valeur détermine la langue à afficher. Si la langue ne correspond pas à une langue définie de PElectro alors l'Anglais est la langue par défaut.
Le retour du message est 1.

```
WM_LEDONOFF= WM_USER+2;
```

Le message WM_LEDONOFF permet d'éteindre ou d'allumer le LED dans la fenêtre des contrôles. WParam à 0, est une demande d'éteindre le LED. Remarque: le LED s'éteint si le nombre de demandes d'éteindre est supérieur ou égal au nombre de demandes d'allumer. Si une demande d'allumer le LED arrive alors le LED s'allume même si le nombre de demande d'éteindre auparavant lui a été supérieur.
Le retour vaut 1.

```
WM_CLEARMES= WM_USER+3;
```

Le message WM_CLEARMES permet d'initialiser le fichier LOG (PElectro.Log) ou la liste des messages dans la fenêtre des contrôles. Si WParam est à 1 le fichier LOG est réinitialisé, sinon, c'est la liste des messages dans la fenêtre des contrôles.
Le retour vaut 1 s'il n'y a pas d'erreur.

```
WM_ADDMES  = WM_USER+4;
```

Ce message permet d'ajouter, au fichier LOG et/ou à la liste des messages de la fenêtre contrôles, les messages provenant du fonctionnement des DLLs. L'adresse du message à ajouter est passée sous la forme d'un PCHAR, dans la variable WParam. LParam à 0, enregistre le message dans la liste et le fichier, à 1 dans la liste et à 2 dans le fichier LOG. Les valeurs supérieures à 2 sont considérées comme étant un LParam à 0 si la DLL ParamTools n'est pas présente. Sinon, le message dont LParam vaut 3, est affiché sous la forme d'un bulle, suivant les paramètres sélectionnés dans la DLL ParamTools, puis le message est enregistré dans le fichier LOG.

```
WM_ALLOCMEM= WM_USER+5;
```

PElectro réserve pour la DLL une zone de la mémoire en nombre d'octets qui sera transmis sous la forme d'un pointeur.
Si WParam<1: le pointeur est à nil.
Si WParam>=1: le pointeur correspond à WParam d'octets.

Exemple: P:= SendMessage(Hdle,WM_ALLOCMEM,100,0); retourne une adresse pointant sur 100 octets dans P.

```
WM_FREEMEM = WM_USER+6;
```

Libère la mémoire réservée. Le pointeur doit être dans WParam.
Si WParam est à nil le retour vaut 0.
Si WParam est libéré sans problème le retour vaut 1, sinon il vaut -1.

```
WM_SAVE    = WM_USER+7;
```

Ce message permet de demander l'enregistrement du circuit en cours à la sortie de PElectro ou lors du passage à un autre schéma de circuit.
Si WParam est à 0, l'enregistrement sera demandé.

```
WM_PICPELEC= WM_USER+8;
```

Un exemple d'utilisation de ce message peut s'effectuer de la manière suivante:

```
Pic := TBitmap(SendMessage(Hdle,WM_PICPELEC,IPosDroit,0));
if Pic<>nil then begin
    PElecCompo.Canvas.Draw(200,100,Pic);
    Pic.Free;
end;
```

Où
Hdle est l'handle de la fenêtre PElectro.
IPosDroit est la valeur d'un indice du tableau PTAc (voir plus bas) dont l'image du composant doit être récupérée.
Pic est un pointeur sur l'objet TBitmap qui contiendra l'image à recevoir. Il s'agit de l'image du composant IposDroit qui est affichée sur le circuit de PElectro.
Si une erreur est survenue, Pic contient la valeur nil.

```
WM_ECHELLE = WM_USER+9;
```

Exemple d'utilisation de WM_ECHELLE.

```
for i:=low(TabDll) to high(TabDll) do
  if PTabDll[i].Name='CiDess.dll' then
    SendMessage(Hdle,WM_Echelle,PTabDll[i].Handle,EchCiDess.ItemIndex);
```

PTabDll[i].Handle est l'handle de la DLL qui a demandée de gérer l'échelle. Ce Handle doit être mis dans WParam).
EchCiDess.ItemIndex (LParam) est un exemple du type d'échelle qui sera affiché par la fenêtre contrôles via la DLL (voir fonction TEchelle).
Le retour du message est toujours 1.

```
WM_HELPPE  = WM_USER+10;
```

Permet d'appeler l'aide (page web) de la fiche de la DLL sur laquelle l'utilisateur a invoqué la touche fonction F1.

Exemple d'utilisation de WM_HELPPE.

```
function TMesure.FormHelp(Command: Word; Data: Integer;var CallHelp: Boolean): Boolean;
var
  i,j: integer;
begin
  i :=0;
  repeat
```

```
    inc(i);
  until (lowercase(PDDllF^.PD^[i].Name)='mesuosci.dll');
  j :=NoLg*1000+0;
  SendMessage(Hdle,WM_HelpPE,j,i);
  CallHelp:=false;
  result:=false;
end;
```

i doit être l'indice de la DLL, dans la table des DLLs, qui demande l'affichage de l'aide. Le nom de la DLL est ensuite utilisé pour la recherche de la page web.
j doit être l'indice de la page web (.htm) de l'aide. Cet indice peut tenir compte de la langue comme dans l'exemple ci-dessus ou toute autre information entière. Une demande au webmaster du site PElectro est nécessaire pour prendre en compte votre page.
Ici le résultat de la page d'aide sera 'http://aide.jbdevelop.com/MesuOsci12000.htm'. Où 'MesuOsci' est tiré de i et '12000' provient de j (dans le cas de la langue française). S'il y a plusieurs fenêtres dans la DLL, la première peut porter le numéro 12000 puis la seconde 12001, etc. Soit, 'MesuOsci12000.htm' puis 'MesuOsci12001.htm' etc.

```
  WM_ALLOCOMP   = WM_USER+11;
```

Permet de réserver à la fin de la table des composants AComposant, le Nombre d'enregistrements défini dans WParam.

```
 WM_PROGRBAR   = WM_USER+12;
```

Ce message permet de gérer depuis une DLL la barre de progression de PElectro.
WParamHi doit contenir la valeur 0 ou 1.
Si WParamHi est à 0, LParamLo contient la valeur minimale de la barre de progression. LParamHi, la valeur maximale et enfin WParamLo le pas (StepBy). La position est mise à 0 lors de ce message.
Si WParamHi est à 1, WParamLo peut prendre les valeurs 1,2 ou 3. Si égal à 1, la progression de la barre est le pas suivant (StepIt). Si à 2, LParamLo contient la valeur de la nouvelle position (Position). Pour la valeur 3, la jauge s'incrémente du pas indiqué dans LParamLo.

```
 WM_STATUBAR   = WM_USER+13;
```

Ce message affiche, dans la statut barre de PElectro, la chaine (pointeur) qui est indiquée dans WParam.

```
 WM_PELCLOAD   = WM_USER+14;
```

Renvoi au programme qui le demande par MessageSys (voir ci-dessous), WParam à 1 et LParam avec le handle de la fenêtre Principale de PElectro si celle-ci est active. Si le programme PElectro se ferme, alors WParam vaut 0.

```
  WM_SIZEXY   = WM_USER+15;
```

Ce message change les dimensions de la surface du circuit.
WParam contient la longueur de l'axe des X.
LParam contient la largeur de l'axe des Y.
Les valeurs maximales de X et de Y sont 3745 pixels.
Le retour du message est normalement 0.

```
  WM_APPTIME   = WM_USER+16;
```

Ce message permet la communication et l'exécution de la DLL « ToolStart » qui gère les utilitaires de l'application PElectro. Le programme de mise à jour de PElectro reçoit les informations de la DLL (Nom, date, heure etc..) qui serviront à lancer les utilitaires demandés. Si un utilitaire est lancé en différé, il faut que le programme de mise à jour (« MAJPElec ») soit actif, sinon un message d'anomalie est envoyé à l'utilisateur.

Les constantes suivantes permettent au programme PElectro d'afficher les informations des DLLs dans les parties visuels du programme. Ces constantes sont à utiliser dans la fonction TGetTypOp. Remarque une DLL ne doit pas à la fois utiliser les constantes Composant et Instrument.

```
Plugins    = $0001;
```

Affiche dans le menu Plugins le nom choisi par la DLL et indique à PElectro que la DLL sait gérer les appels correspondants (voir fonctions ou procédures expliquées ci-dessous).

```
Composant  = $0002;
```

Permet au programme PElectro de savoir que la DLL gère des composants (PElectro affiche les composants dans la fenêtre adéquate via les procédures et fonctions de la DLL).

```
Instrument = $0004;
```

Indique que la DLL correspond à un instrument Elc (affichage dans la fenêtre des instruments).

```
Parameter  = $0008;
```

La DLL doit apparaître dans la liste des paramètres de PElectro (menu Paramètres->Options générales)

```
MenuPopUp  = $0010;
```

Dans le menu PopUp (clic droit sur le circuit) figurera le nom désigné par la DLL. La DLL devra gérer les procédures ou fonctions correspondantes (voir plus loin procédures et fonctions).

```
Tools  = $0020;
```

Seule la première boîte à outils qui utilise cette valeur dans la liste des DLLs pourra communiquer avec le programme PElectro via les procédures AltPElec et MesPElec (Voir procédures et fonctions ci-dessous).

```
OpenMenu  = $0040;
```

Rajoute dans le menu 'Fichier->Ouvrir autres formats', Une ligne de Menu définie par la DLL.

```
InstruBox  = $0080;
```

Indique que la DLL correspond à un instrument Elp (affichage dans la fenêtre des instruments).

```
Generator  = $0100;
```

Indique que la DLL est un instrument de type « Générateur de courant ou de tension ».

Les paramètres ci-dessous peuvent être utilisés avec le message WM_LOCKFUNC. Ils doivent être transmis dans le champ WParam du message.

```
LockNew    = $0001;
```

LockNew permet de verrouiller/déverrouiller les fonctions de passage à un circuit vierge.

```
LockOpen   = $0002;
```

LockOpen permet de verrouiller/déverrouiller le passage à un autre circuit.

```
LockAffi   = $0004;
```

LockAffi permet de verrouiller/déverrouiller les fonctions d'affichage du circuit.

```
LockCCP    = $0008;
```

LockCCP permet de verrouiller/déverrouiller les fonctions du presse papier.

```
LockNComp  = $0010;
```

Permet d'empêcher la mise en place sur le schéma d'un nouveau composant.

```
PixelMax = 32768;
```

Taille maximale d'une image Bitmap gérable par PElectro.

```
cstMaxChars = 5000; // petite taille tampon chaîne
cstBigMaxChars = 300000 // grande taille de chaîne
```

Les tailles maximales des chaînes de caractères passées entre PElectro et les DLLs suivant la fonction concernée.

```
Defaut   = $00;

Francais = $0C;
English  = $09;
Arabe    = $01;
Chinois  = $04;
Allemand = $07;
Hebreu   = $0D;
Japonais = $11;
Italien  = $10;
Russe    = $19;
Espagnol = $0A;
Hindi    = $39;
Portugais= $16;
Indonesien=$21;
```

Valeur des langues disponibles dans PElectro.

```
CstWH00 = 780174372;
```

Valeur des composants de la DLL WH00 nécessaire au fonctionnement de PElectro (voir TabLibDll. Ci-dessous). Cette DLL est toujours en première position dans la table TabLibDll.

```
VersAct = 6;
```

Version actuelle des fichiers de PElectro

```
NbrMaxDll= 200;
```

C'est le nombre maximum de DLLs qui peut être utilisé par PElectro.

```
NbrParDll= 30;
```

C'est le nombre de paramètres gérable par la DLL. Les paramètres de 0 à 24 sont réservés pour PElectro. Chaque DLL dont la constante est Parameter doit gérer NbrParDll paramètres par la fonction TCopyParam (voir procédures et fonctions plus loin). Exemple, la première DLL définie par TGetTypOp à Parameter devra au maximum gérer les paramètres de 25 à 49 et ainsi de suite pour les autres DLLs.
Les paramètres de 0 à 22 sont les suivants:
0 = la température.
1 = le temps par divisions.
2 = le sens du courant.
3 = le nombre d'itérations.
4 = la position.
5 = la précision.
6,7 = top et left des boîtes à outils.
8 = le rang de l'imprimante dans la liste des imprimantes.
9 = si le journal est activé.
10= si automatique activé.
11= si les bulles sont activées.
12= la largeur du circuit.
13= la hauteur du circuit.
14= le numéro de langue en cours d'utilisation.
15= si le magnétisme sur le circuit est activé.
16= si le pas sur le circuit est visible.
17= si le pas de 16 pixels sur le circuit est activé.
18 = L'écriture en Unicode est possible dans CiDess si le paramètre est à 1 ou Ansi si le paramètre est à 0.
19 = Le mode temporel (0), fréquentiel (1) ou polarisation (2).
20 = Le nombre de paramètres par DLL.
21 = le Nombre de DLLs paramètres.
22 = Le type de circuit, Elc = 1, Elp = 2 et CDP = 3.
23 = Le zoom utilisé.
24 = La largeur des pistes du circuit.
25 = La distance entre les pistes.

```
NbrOff= 1E37;
```

```
MessagePElectro = 'PElectroJBa'
```

Permet d'envoyer un message particulier à PElectro à partir d'une autre application.
La gestion des messages est la suivante:
-D'abord l'application, qui veut interroger PElectro, enregistre ce message.
-Puis envoi, par SendMessage, le handle de la fenêtre de l'application dans LParam et 1 dans WParam.
-Si PElectro est actif, il renvoie le message 'PElcLoad' avec 1 dans WParam et le handle de la fenêtre principale de PElectro dans LParam, puis lors de sa fermeture WParam passe à 0.

Cette séquence offre à une application extérieure le moyen de savoir si PElectro est actif puis s'il s'est fermé.

```
MessageLangue = 'LanguageJBa'
```

Permet à PElectro d'envoyer un message indiquant aux autres applications que la langue a changée.
La gestion du message est la suivante:
-D'abord l'application, qui veut recevoir la langue utilisée par PElectro, enregistre ce message.
-Puis lorsque la langue change, la nouvelle langue ce trouve dans le paramètre WParam du message.
L'application réceptrice peut alors s'adapter.

Structures PElectro

```
Type
  TA = array of array of double ;
  TI = array of double ;
  PTA = ^TA ;
  PTI = ^TI ;
```

TNotCompo permet de limiter les fonctionnalités de PElectro ainsi que des éléments du menu PopUp (lors du clic droit sur le circuit de PElectro). Cette énumération fonctionne avec le champ RInterdit du record AComposant (voir plus loin).

```
  TNotCompo=(
        jbOpen,
```

Dans PElectro, l'ouverture ne peut se faire que dans le cadre d'une DLL Instrument. jbOpen interdit l'ouverture de l'instrument en grisant l'item du menu correspondant.
Exemple: PA[i].RInterdit:=PA[i].RInterdit+[jbOpen];

```
        jbMove,
```

Interdit le déplacement du composant ou de l'instrument.

```
        jbRota,
```

Interdit la rotation du composant ou de l'instrument.

```
        jbConnect,
```

Interdit la liaison du composant ou de l'instrument avec un autre.

```
        jbSeparate,
```

Interdit la suppression d'une liaison du composant ou de l'instrument.

```
        jbDelete,
```

Interdit la suppression du composant ou de le couper.

```
        jbOne,
```

Si un autre instrument de la même DLL que celui en cours (clic droit sur le composant) contient cet indicateur alors un seul de ces composants peut être ouvert.

```
        jbInvers,
```

Interdit l'inversion du composant ou de l'instrument.

```
        jbOther
```

Interdit à un instrument ou à un composant de se lier à celui ayant cet indicateur.

```
        );
```

```
TTypeCompoDonVisual = record
  Min,Max: double;
  Couleur: TColor;
  Taux    : integer;
  Ecart   : double;
  TypeCtrl: word;
  OutValL: boolean;
  TypeOpC: array [0..30] of Char;
  LogOnOff: boolean;
End;
```

Cinq types de composants 'TypeCtrl' peuvent s'afficher dans la fenêtre du plugin GestCompo.

1- **boîtes à cocher.**
2- **LED.**
3- **boutons rotatifs.**
4- **Display.**
5- **jauge.**

<u>Les boîtes à cocher:</u>
Si 'Max = 0' la case à cocher est décochée. Sinon elle est cochée.
La boîte à cocher prend la couleur 'Couleur'.
Ce composant est modifiable en temps réel et informe la DLL correspondante des changements.

<u>Les LED :</u>
Ce composant est en affichage.
La LED prend la couleur 'Couleur' si elle est allumée.

<u>Les Boutons rotatifs :</u>
Ce composant est modifiable en temps réel et informe la DLL correspondante des changements.
La couleur du composant est donnée par 'Couleur'.
Si 'Ecart = 0' l'écart entre deux positions est calculé de la manière suivante, Ecart = '(Max – Min) / 100'. Si 'Taux = 0' La position de départ est au milieu. Sinon la position de départ est égale à 'Taux'.
Si 'Ecart <> 0' la position de départ est 'Min'.
Si 'LogOnOff' est à false, la position maximale est égale à '(Max – Min) / Ecart'. Si 'LogOnOff est à true, alors le calcul est égal à 10 à la puissance (Position du bouton * Ecart * Max / position maximale du bouton).
Si 'OutValL' est à true, alors la valeur du composant s'affiche avec le multiplicateur adéquate (exemple : µ pour micro, p pour pico, etc.….).
'TypeOpC' peut contenir une chaine qui sera rajoutée après la valeur du composant (exemple : Ohm).

<u>Les Display :</u>
La couleur d'affichage est 'Couleur' (segments).
'LogOnOff' précise si un séparateur apparait (true) ou non (false).

<u>Les jauges:</u>
La plage de fonctionnement de la jauge est de 0 à 100.
La couleur de la jauge est donnée par 'Couleur'.
Si 'LogOnOff' est à true, la jauge est verticale sinon elle est horizontale.
Si 'Taux' est différent de zéro, la largeur ou l'épaisseur de la jauge est définie par 'Taux'.
'Max' défini la position de la jauge au départ.
Au cour du calcul, si 'Min' est différent de zéro et que la position est inférieure à 'Min', la couleur de la jauge devient rouge.

{ Données des DLL jointes aux Schémas }

```
TDonDll = record
```

TDonDll donne la possibilité à la DLL d'avoir dans le composant de son choix (AComposant) une zone mémoire sauvegardée avec le circuit.

```
  Composant:longword;
```

C'est le numéro de DLL calculé par PElectro.

```
  LgDonDll: word;
```

Taille de la zone réservée par la DLL.

```
  DnDonDll: pointer;
 end;
```

Pointeur sur la zone réservée.

{ Définition des bornes d'un composant }

```
  TBorne = record
    PosX: SmallInt;         { Position X borne / Top bitmap }
    PosY: SmallInt;         { Position Y borne /  Top Bitmap }
```

Position x et y de la borne par rapport au point RX et RY de l'image du composant. Ces deux zones ne sont pas utilisées dans l'enregistrement RComposant.

```
    Bi: longword;           { Composant type I attaché }
    BR: word;               { rang du composant attaché }
    BVal: byte;             { Borne composant attaché }
    BBtr: byte;             { Equivalent borne réelle d'un boîtier constructeur }
```

Bi est le RComposant du composant lié.
BR est le RangCompo du composant lié.
BVal est la borne du composant lié.
BBtr est le numéro de la borne réelle du composant dans un boîtier si différent de BVal.

```
  end;
```

```
  TLigne = record
    PosX: SmallInt;         { position x du lien }
    PosY: SmallInt;         { position y du lien }
```

PosX et PosY sont les positions d'un segment de départ ou d'arrivé parmi les segments formant la liaison. PosX et PosY sont à zéros lorsque la liaison est terminée.

```
  End;
```

{ Définition d'un composant à afficher }

```
  RComposant = record
```

RComposant est une structure enregistrement (record) d'un des éléments qui constitue le composant proprement dit (AComposant). C'est à la DLL de fabriquer les enregistrements formant le composant AComposant. Voir

procédures et fonctions.

```
RComposant: longword;  { Composant I }
```

Il doit contenir le numéro calculé par PElectro de la DLL composant ou instrument. Tous les composants contenus dans la DLL portent ce numéro. Voir par exemple le numéro de la DLL WH00 ci-dessus (CstWH00).

```
RNumCompo: word;       { Composant J de I}
```

Ce champ représente le numéro d'ordre des composants dans la DLL définie par RComposant.

```
AffiCompo: integer;    { A quel composant il est lié }
```

Il contient l'indice correspondant au composant AComposant dont il est un élément.

```
LienCompo: integer;    { Lien avec un autre composant }
```

Lien avec un autre composant. LienCompo est la différence entre l'indice de l'autre RComposant lié et l'indice du RComposant qui le lie. Si positif, le lieur se trouve avant le lié. Si négatif, le lieur se trouve après le lié. Si nul pas de lien. Si 999999 une erreur s'est produite, il faut ressaisir le composant à relié.

```
RangCompo: word;       { No du composant dans sa classe I }
```

Le rang doit s'incrémenter de 1 à chaque nouvel enregistrement. Voir procédures et fonctions.

```
RNbBorn: word;         { Nbre de borne du composant }
```

Peut prendre comme valeur 1 ou 2. voir RBorne ci-dessous. Ne peuvent figurer dans les enregistrements RComposant que des composants ayant une borne (exemple la masse) ou deux (par exemple une résistance, un condensateur, un interrupteur, etc...)

```
RBorne: array[0..1] of TBorne;   { PosX et PosY peuvent servir de tag }
```

L'indice 0 correspond à la borne 0 du composant (exemple pour une résistance dans la DLL WH00, l'indice correspond au haut de celle-ci par rapport au point rouge de l'image du composant) et 1 à l'autre borne (soit pour la résistance, au bas du composant).

```
RABorn: array[0..1] of SmallInt; { Borne du composant affiché }
```

Si la borne 0 ou 1 ne correspond à aucune borne relier à l'extérieur (défini par l'enregistrement AComposant) ou plus généralement si la borne est reliée à un autre composant interne (même AffiCompo). Le composant RComposant doit avoir la valeur -1 dans RABorn[x] et les éléments de RComposant.RBorne doivent être renseignés. Sinon il doit comporter l'indice d'une des bornes de AComposant.RBorne. Exemple un transistor aura 3 bornes 0,1 ou 2 correspondant au collecteur, base, émetteur, il faudra que les composants interne reliés à l'extérieur ait la valeur 0, 1 ou 2 dans RABorn[x]. Dans ce cas RComposant.RBorne[0] ou [1] sera automatiquement rempli pas PElectro lorsque l'utilisateur reliera les composants AComposant entre eux. Remarque: Plusieurs composants interne de AComposant ne peuvent pas utiliser le même indice dans RABorn[x], sinon le composant AComposant ne fonctionnera pas.

```
Sens: shortint;     { Place réservée }
Rien1: array[0..2] of byte;     { Place réservée }
Rien2: integer;     { Place réservée }

Indice: SmallInt;      { Rang tableau courant / tension voir (ZeroPot) }
```

Utilisé pour la simulation (exemple oscilloscope) de diverses grandeurs comme la tension d'un composant. Voir PDF correspondant.

```
  Rien3:  array[0..6] of word;     { Place réservée }

  RDonne: array[0..9] of double;   { données du composant }
end;
```

Un sous composant RComposant d'un composant AComposant peut avoir au maximum 10 paramètres (de 0 à 9). Mais AComposant peut avoir plusieurs RComposant, les paramètres de AComposant (de 0 à 19) peuvent donc être répartis sur plusieurs RComposant. D'autre-part La DLL peut réserver un pointeur RPtr (ci-dessous) pour mettre d'autres valeurs en cas d'utilisation d'une base de données avec beaucoup de paramètres.

{ Liste des composants à afficher }

```
  AComposant = record
```

AComposant contient les informations qui serviront au fonctionnement de PElectro comme par exemple l'affichage du composant qui apparaîtra dans la fenêtre circuit.

```
  RComposant: longword;  { Composant I }
```

Il contient le numéro calculé par PElectro de la DLL composant ou instrument. Tous les composants contenus dans la DLL portent automatiquement ce numéro. Voir le numéro de la DLL WH00 ci-dessus (CstWH00).

```
  RNumCompo: word;       { Composant J de I}
```

Ce champ représente l'ordre établi des composants dans la DLL.

```
  RTypCompo: array[0..9] of Char;  { Caractère identifiant le composant }
```

Il contient les caractères identifiant la DLL des composants (exemple Std, Res, Ins) qui devront être différents d'une DLL à une autre. Les caractères sont codés en Unicode.

```
  RX: integer;           { Top X }
  RY: integer;           { Top Y }
```

RX et RY sont les coordonnées de référence du haut gauche du composant.

```
  BX: word;              { bas X droit }
  BY: word;              { bas Y droit }
```

BX et BY sont les coordonnées de référence du bas droit du composant par rapport à RX et RY.

```
  PageCompo: word;        { Place réservée }
  Rien3: word;        { Place réservée }

  RangCompo: word;       { No du composant dans sa classe I }
```

Rang successif des composants pour une même DLL. Le rang est calculé par PElectro à chaque nouveau composant affiché dans la fenêtre circuit. Il apparaît dans le nom du composant affiché (exemple: Std2, Ins1 etc.)

```
  RotationA: SmallInt;   { Angle de rotation }
```

Valeur utilisée par PElectro pour gérer les rotations.

```
  RInterdit: set of TNotCompo;{ Interdire une fonction }
```

Voir TNotCompo plus haut.

```
  RNbBorn: word;         { Nbre de borne du composant }
```

C'est le nombre de bornes du composant de 1 à 61. Voir RBorne.

```
  RBorne: array[0..60] of TBorne;
```

Tableau de 61 bornes maximum. Voir TBorne.

```
  RBornS: array[0..60,0..19] of TLigne;
```

Segments (20 maximum) qui composent une liaison d'une des bornes pour un maximum de 61 bornes.

```
  Rien2: smallint;              { Place réservée }

  ADonne: array[0..19,0..15] of Char; { données du composant }
```

Il est possible sans base de données par exemple d'avoir pour un composant 20 paramètres (fenêtre Valeurs de PElectro) qui décriront celui-ci. Exemple: La résistance, la température, la capacité, etc.…). Tous les types simples de paramètres sont acceptés (réel, booléen, entier et texte). C'est à la DLL de tester et de corriger les erreurs de frappe ou de valeur (voir procédures et fonctions). Ensuite la DLL charge dans les constituants du composant les valeurs à leur bonne place dans RDonne (10 paramètres maximum, voir ci-dessus). Les caractères sont des caractères Unicode.

```
  ADonDll: array[0..19] of TDonDll; { données des DLL }
```

20 DLLs peuvent alimenter cette table de données qui est sauvegardée avec le circuit.

```
  RInvers: SmallInt;     { Inversion du Bitmap; -1=inversion }
```

Indique si l'inversion du composant a été demandée.

```
  PosText: byte;           { Position du texte }
```

Cette valeur place le texte du composant:
- **0, affiche le texte à droite.**
- **1, affiche le texte au centre.**
- **2, affiche le texte en dessous.**
- **3, affiche le texte à droite et en dessous**
- **Une autre valeur, fonctionne comme 1.**

```
  TempTag: byte;           { Place réservée }

  RPtrCompo: pointer;      { Pointeur pour sauvegarde données DLL }
```

Ce pointeur sur une zone mémoire ne peut être utilisé que par la DLL qui fait référence au composant de cet enregistrement.

```
  TypeEnvCtrl: double;
  TypeRecCtrl: double;
```

Ces deux zones servent à la DLL GestCompo, à envoyer ou à recevoir des données aux DLLs. L'aide de GestCompo.dll est sur le site 'http://www.jbdevelop.com/'.

```
  LgPtr: word;        { longueur zone RPtrCompo }
```

Longueur de la zone RPtrCompo.

```
  RLgPtr: word;        { longueur zone RPtr }
  RPtr: pointer;       { Pointer sur une structure }
```

PElectro fournit à chaque composant AComposant une zone mémoire demandée au préalable par la DLL via la fonction TAreaCompo.
RLgPtr contient le nombre d'octets sur lesquels pointe RPtr.
Exemple:

```
function AreaCompo(h:THandle;var D:AComposant;var TBDll:array of TabLibDll; Pa:PTPa):integer; stdcall;
begin
  result := sizeof(TZonePar);
end;
```

Dans la fonction MakeCompo:

```
    A[j1].RLgPtr := sizeof(TZonePar);
    A[j1].RPtr := P;
  end;
```

```
  PTAc = array of AComposant;
  PTRc = array of RComposant;
  PPTa = ^PTAc;
  PPTr = ^PTRc;
```

Les tables des composants et des sous-composants n'ont pas de taille définie.

```
 PTPa = array of double;
 PPTp = ^PTPa;
```

La taille de la table de paramètres dépend du nombre de DLLs.

{ Table des DLLs }
L'enregistrement ci-dessous regroupe les informations recueillies par PElectro pour une DLL donnée.

```
 TabLibDll = record

  Name: array[0..16] of Char;
```

Ici se trouve le nom de la DLL chargée. Les caractères sont codés en Unicode.

```
  Handle: HModule;
```

Contient le Handle de la DLL. Celui-ci servira à l'accès aux procédures de la DLL.

```
  Composant: longword;
```

Composant est calculé par PElectro en fonction du nom de la DLL. Il est utilisé pour tous les composants de la DLL.

```
  Operat: word;
```

Ici se trouvent toutes les constantes d'opération de la DLL comme Parameter, Plugins, etc.…

```
  RangCp: word;
```

Cette zone est utilisée par PElectro.

```
  Divers: byte;
```

La DLL peut spécifier l'un des types suivants:
= 0 si la DLL n'active pas de fenêtre dans la fonction TChangeCompo.
= 1 si la fonction TCalCompo contient un traitement.
= 2 si une fonction de la DLL appelle une fonction TCalCompo d'une autre DLL.
Chaque type est exclusif. Une DLL ne peut être de type 1 et 2 car PElectro ne fonctionnera pas.

```
  Activate: byte;
```

Cette zone est utilisée par PElectro pour compter le nombre d'instances actives correspondant à cette DLL. Elle permet, en liaison avec le champ « Divers », d'autoriser le lancement d'une instance d'une autre DLL.
La fonction de lancement de « TchangeCompo » est implémentée de la façon suivante dans le programme principal PElectro:

```
function LancChangeCompo(h: THandle; Activate: Boolean = false): Boolean;
var
  i, j: Integer;
  k: byte;
begin
  { Lancement ? }
  result := false;
  with Composants do
  begin
    k := 0;
    for i := 0 to high(VTabDll) do
    begin
      if VTabDll[i].Handle = 0 then
      begin
        raise EDLLError.Create('Error VTabDll ' + IntToStr(h))
          at@TFenMain.OnExceptionErr;
        exit;
      end;
      if VTabDll[i].Handle = h then
      begin
        k := VTabDll[i].Divers;
        break;
      end;
    end;
    for j := 0 to high(VTabDll) do
    begin
      if k = 0 then
        break;
      if VTabDll[j].Handle = 0 then
        break;
      if VTabDll[j].Divers = 0 then
        continue;
      if (k <> VTabDll[j].Divers) and (VTabDll[j].Activate > 0) then
        exit;
    end;
    if Activate then
      VTabDll[i].Activate := VTabDll[i].Activate + 1;
  end;
  result := true;
end;
```

Si le retour de la fonction est false, l'instance de la DLL n'est pas lancée.

```
  NbrMaxCompo: word;
```

Correspond au nombre maximum de composants de la DLL pouvant figurer sur le schéma en cours (fonction TGetMaxCompo).

```
    NbrCompo: word;
  end;
```

Il indique le nombre de composants de la DLL dans le circuit en cours.

```
 TabDll = array of TabLibDll;
 PTabd  = ^TabDll;
```

200 DLLs peuvent être chargées par PElectro dans cette table.

Cette zone est utilisée conjointement avec la fonction TCalOpera (voir ci-dessous).

```
  TTabCompoOp = record
    R: double;
```

Il s'agit de l'impédance d'une résistance.

```
    C: double;
```

Il s'agit de la capacitance de la capacité.

```
    L: double;
```

Il s'agit de l'inductance de l'inductance.

```
    E: double;
```

Valeur d'une source de tension.

```
    I: double;
```

Valeur de la source de courant.

```
    TypeLien: word;
```

Valeur du lien,
1 courant/courant
2 courant/tension
3 tension/tension
4 tension/courant

```
    // 2 composants pouvant être liés
    Beta1: double;  { facteur de TypeLien }
    Lien1: integer; { Composant lié }
    Beta2: double;  { facteur 2 de TypeLien }
    Lien2: integer; { Composant 2 lié }
  end;
```

Beta1 et Beta2 sont les coefficients du lien Lien1 et Lien2 respectivement. Si ces zones ne sont pas utilisées, Beta1/Beta2 doivent être à 0 et les liens Lien1/Lien2 à -1.

Fonctions PElectro

```
TGetTypOp =  function:word; stdcall;
```

Cette fonction doit être présente dans la DLL.
Voici des exemples de fonction GetTypOp pour les DLLs.

```
function GetTypOp: word; stdcall;
begin
  result:=Composant+Plugins;
end;
```

La DLL informe PElectro qu'il s'agit d'une DLL de composants et qu'elle doit figurer dans le menu Plugins.

```
function GetTypOp: word; stdcall;
begin
  Result := (Parameter or MenuPopUp or Plugins);
end;
```

Les possibilités de cette DLL est d'apparaître dans les menus Plugins, paramètres et dans Popup de PElectro.
D'autres procédures / fonctions seront nécessaires pour qu'elle puisse fonctionner correctement.

```
TMemDonDll=  function:integer; stdcall;
```

Cette fonction peut être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
TMemDonDll permet de réserver par PElectro une zone mémoire dont la longueur est donnée par le retour de cette fonction. Le pointeur qui recevra cette zone est RPtrCompo.

```
TIdentity =  procedure(var D,N,V,C,M: String); stdcall;
```

Cette fonction doit être présente dans la DLL quel que soit la constante qui figure comme opération de la DLL.
Exemple:

```
procedure Identity(var D,N,V,C,M: String); stdcall;
begin
  D := 'ElcAop';
  N := 'JBa';
  V := 'V1.0';
  C := 'www.jbdevelop.com';
 M := 'Composant Ampli Op';
end;
```

D est le nom de la DLL.
N l'auteur.
V la version.
C éventuellement le site web.
M une phrase décrivant le contenu de la DLL.

```
TGetCompo =  procedure(ListCompo:PChar; n:integer; NoLg:word; NoList: word = 0); stdcall;
```

Cette procédure doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Cette procédure permet de transmettre les noms des composants et le nom de la bibliothèque par ListCompo.
Le nom de la bibliothèque doit être précédé par le caractère 'P' et se terminer par le caractère ';'.
Chaque nom de composant doit commencer par le caractère 'I' et se terminer par le caractère ';'.
n est le nombre de caractères maximum+1 admissible par ListCompo.
Exemple de ListCompo:
'Ptransistors;I2N2222;I2N2229; etc…….'.
NoLg contient le numéro de langue (exemple $0C pour le français, voir plus haut).
Pour les DLLs instruments seulement le nom doit figurer dans ListCompo. Exemple 'Oscilloscope;'.

```
TAreaCompo=  function(H:HWND; A: AComposant; var TBDll: TabDll;var Parm: PTPa):integer; stdcall;
```

Cette fonction doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
La fonction TAreaCompo renvoie la taille de la zone mémoire qui sera atteignable par le pointeur RPtr.
Exemple:

```
function AreaCompo(h:HWND;var D:AComposant;var TBDll:TabDll;
                Pa:PTPa):integer; stdcall;
begin
  result := sizeof(TZonePar);
end;
```

```
  TRInCompo = TAreaCompo;
```

Cette fonction permet de fournir le nombre maximum à réserver pour les sous composants utilisés par la DLL.
Exemple:

```
function RInCompo(h:HWND;var D:AComposant;var TBDll:TabDll;
                Pa:PTPa):integer; stdcall;
begin
  result := 15;
end;
```

```
  TMakeCompo=  function(H:HWND; i: word; A:PTAc;var TBDll: TabDll;var Parm: PTPa; P:pointer;
NoLg:word):boolean; stdcall;
```

Cette fonction doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Cette fonction permet de créer le composant AComposant.
Les paramètres:
H correspond au handle de l'application PElectro.
i est l'indice de l'endroit où l'AComposant doit être mis à jour.
A est un pointeur sur la table des AComposant.
TBDll est le tableau des informations relatives aux DLLs.
Parm un pointeur sur la table des paramètres.
P un pointeur sur la zone réservée par PElectro pour un composant AComposant.
NoLg est le Numéro de langue.
Si la fonction s'est bien terminée, le retour de la fonction doit être à true.

```
  TInitFlag =  procedure(var A: AComposant); stdcall;
```

Cette procédure doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Permet d'initialiser les indicateurs d'un composant AComposant.
Exemple:

```
procedure InitFlag(var D:AComposant); stdcall;
begin
  D.RInterdit := [jbInvers,jbOther];
end;
```

```
  TChangeCompo=function(H:HWND; I,ACount: word;var A:PTAc;var TBDll: TabDll;var Parm: PTPa;
NoLg:word):boolean; stdcall;
```

Cette fonction doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Cette fonction permet de mettre à jour le composant AComposant si la DLL est de type composants.
Cette fonction pour une DLL Instrument permet de lancer l'ouverture de l'instrument (affichage d'une fiche).
Les paramètres:
H correspond au handle de l'application PElectro.
i est l'indice de l'endroit où est l'AComposant.
ACount le nombre d'élément dans la table contenant les AComposants.

A est la table des AComposant.
TBDll est le tableau des informations relatives aux DLLs.
Parm la table des paramètres.
NoLg le numéro de langue en cours.
Si la fonction s'est bien terminée, le retour de la fonction doit être à false.

```
TOpenCompo=function(H:HWND; i: word;var A:PTAc;var TBDll: TabDll; Parm: PTPa;
NoLg:word):boolean; stdcall;
```

Cette fonction peut être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Cette fonction permet de mettre à jour les composants AComposant à l'ouverture d'un nouveau fichier '.Elc'.
Par exemple ceux qui utilisent une base de données pourront mettre à jour les circuits qui contiennent leurs références en cas de changement de paramètres de la base de données (Par le Plugins DataBase éventuellement).
Cette fonction si elle est présente appellera généralement la fonction TchangeCompo, ci-dessus, s'il s'agit d'une DLL de composants.
Les paramètres:
H correspond au handle de l'application PElectro.
i est l'indice de l'endroit où est l'AComposant.
A est un pointeur sur la table des AComposant.
TBDll est le tableau des informations relatives aux DLLs.
Parm un pointeur sur la table des paramètres.
NoLg le numéro de langue.
Si la fonction s'est bien terminée, le retour de la fonction doit être à false.
Exemple de code pour la DLL transistors.

```
function OpenCompo(h:HWND; i:word;var A:PTAc;
                var TBDll: TabDll;
                Pa:PTPa; NoLg:Word):boolean; stdcall;
begin
  result:=ChangeCompo(h,i,i,A,TBDll,Pa,NoLg);
end;
```

```
TPointCompo= procedure(var A: AComposant); stdcall;
```

Cette procédure doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Permet de mettre à jour les informations de AComposant. Celles qui sont obligatoires dans cette routine sont:
A.BX et BY aux dimensions du bitmap.
A.RNbBorn le nombre de bornes.
A.RBorne en fonction du nombre indiqué par A.RNbBorn -1.

```
TDonneCompo= procedure(var h:HBITMAP;var A:AComposant; Update:Boolean); stdcall;
```

Cette procédure doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
PElectro par l'indicateur Update demande à la procédure de réinitialiser les données définies dans TPointCompo.
D'autre part PElectro veut recevoir l'image du composant AComposant qui sera affichée dans la fenêtre Circuit.
Exemple:

```
procedure DonneCompo(var h:HBITMAP;var D:AComposant; Update:boolean); stdcall;
begin
{ info du composant }
  h := LoadBitmap(hInstance,PChar('IMAGC'+IntToStr(D.RNumCompo+2)));
  if Update then PointCompo(D);
end;
```

```
TCtlValCompo=procedure(ListVal:PChar; n,j:integer; NoLg:word;var A:PTAc; iPos,Ni:word); stdcall;
```

Cette procédure doit être présente dans la DLL si la constante Composant figure comme opération de la DLL.

ListVal est une liste en entrée et en sortie.
n est le nombre de caractères maximum+1 de ListVal.
j le jème composants de la DLL. j débute à 0.
NoLg contient le numéro de langue (exemple $0C pour le français, voir plus haut).

ListVal en entrée est la liste des valeurs du composant telles qu'elles apparaissent dans la fenêtre Valeurs de PElectro après la saisie par l'utilisateur. Chaque valeur est séparée par un point-virgule ';'.
La DLL doit contrôler la validité de ces saisies.
ListVal en sortie est la liste des valeurs aussi bien valides que corrigées toujours séparées par un caractère point-virgule ';'.

TImagCompo= procedure(i: integer;var H: HBITMAP; NoList: word = 0); stdcall;

Cette procédure doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Pour les DLLs Composants, i est le ième image des composants de la DLL. Si i est à 0 alors il faut transmettre l'image que l'on veut voir apparaître dans l'onglet de la bibliothèque de la fenêtre des composants.
Exemple de code:

```
procedure ImagCompo(i: integer; var h: HBITMAP); stdcall;
begin
{ Image dans la boîte de composant }
  h := LoadBitmap(hInstance,PChar('IMAG'+IntToStr(i+1)));
end;
```

Si la DLL est de type instrument alors une seule image doit être transmise, celle caractérisant l'instrument.

TGetValCompo=procedure(ListCompo: PChar; n,j: integer; NoLg: word); stdcall;

Cette procédure doit être présente dans la DLL si la constante Composant figure comme opération de la DLL.
ListCompo est une liste en sortie des valeurs par défaut, du composant, séparées par un point-virgule ';'.
n est le nombre de caractères maximum+1 de ListCompo.
j le jème composants de la DLL. j débute à 0.
NoLg contient le numéro de langue (exemple $0C pour le français, voir plus haut).

Une valeur est constituée de la manière suivante:
Le libellé du champ terminé par point-virgule.
La valeur du champ débutant par 's','f','b' ou 'i' (string, float, boolean ou integer) puis le caractère 't' ou 'f' (true ou false pour savoir si la zone est modifiable ou grisée), une chaîne du type de la valeur du champ (exemples si le type est 'i', la chaîne est un integer comme '10'. Si le type est 'f' la chaîne est un float comme '1,2E-5'. Etc..) fini par le caractère ';'.
L'unité de mesure suivie du caractère ';'.
Exemple d'une valeur: 'Taux;it100;%;'. Ce champs sera modifiable.

TgetMaxCompo=function(i: word): shortint; stdcall;

Cette fonction doit être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Le résultat de TGetMaxCompo représente le nombre maximum de composants de la DLL qui peuvent être utilisés sur un circuit. Cela permet de limiter par exemple en démonstration l'utilisation de la DLL.
La valeur 0 signifie que le nombre de composants est illimité.
Exemple:

```
function GetMaxCompo: byte; stdcall;
begin
  result := 0;
end;
```

TCopyParam = function(var parm: PTPa; i: integer):boolean; stdcall;

Cette fonction doit être présente dans la DLL si la constante Parameter figure comme opération de la DLL.

Parm représente la liste des paramètres à mettre à jour. L'indice du premier paramètre est donné par i. Le nombre de paramètres est fixé par la variable NbrParDll par DLL. Le retour à true signifie qu'une erreur s'est produite.
Un exemple d'utilisation pour un seul paramètre géré par la DLL.

```
function CopyParam(var parm: PTPa; i: integer):boolean; stdcall;
var
  F: TMemIniFile;
begin
  F := TMemIniFile.Create('MesuOsci.ini');
  Parm[i] := strtofloat(F.ReadString('General','Taille','010'));
  F.Free;
  result:=false;
end;
```

```
TGetMenuCompo=procedure(LCompo: PChar; n: integer;var h: HBITMAP; NoLg: word); stdcall;
```

Cette procédure doit être présente si la constante MenuPopUp figure dans la DLL.
L'image définie par h sera affichée dans le PopUp de PElectro (clic droit sur un composant du circuit).
LCompo est le libellé qui apparaitra dans l'item du menu PopUp.
n est la longueur maximum+1 de LCompo que peut recevoir PElectro.
NoLg contient le numéro de langue (exemple $0C pour le français, voir plus haut).
Exemple:

```
procedure GetMenuCompo(List:PChar; n:integer;var h:HBITMAP; NoLg:word); stdcall;
begin
  StrLCopy(List,PChar(MesLangID(NoLg,164)),n-1);
  h := LoadBitmap(hInstance,PChar('Image0'));
end;
```

```
TPrintCompo= function(var parm: PTPa; ListCompo: PChar; n,i: integer; NoLg: word; var TBDll: TabDll;
out Pos,Width: double; out Eof :boolean):boolean; stdcall;
```

Cette fonction peut être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Permet d'imprimer un texte dans PElectro lors de la demande d'impression des instruments ou des composants des fenêtres correspondantes.

N, i, Parm, et TBDll est en entrée.
NoLg contient le numéro de langue (exemple $0C pour le français, voir plus haut).
Pos, width et Eof en sortie.
ListCompo en entrée/sortie.
Parm contient les paramètres de PElectro et des DLLs.
TBDll contient les informations sur les DLLs qui ont été chargées par PElectro.
N est la longueur maximum+1 des caractères du champ ListCompo.
i représente la ième itération de la fonction TPrintCompo pour la DLL.
ListCompo est la chaîne de caractères à imprimer.
Pos et Width permettent, si Pos et Width sont >0, d'imprimer ListCompo à la position Pos sur une largueur Width.
Si Width=-1 l'impression se fait sur une ligne.
Si Pos=-1 il s'agit d'un changement de page.
Si Width=0 l'impression d'une ligne blanche s'effectue.
Si Pos=0 une ligne de tirets s'imprime.
Si Pos et Width <-1 l'impression de la 'Pos' image du composant ou de l'instrument depuis ImagCompo s'effectue sur la 'Width' ligne.
Les autres combinaisons possibles de Pos et de Width ne sont pas pour le moment implémentées.
Eof indique la fin de l'impression par la DLL si égal à true.
Exemple:
Si la DLL veut imprimer 11 lignes. i variera de 0 à 9 avec Eof=false. Puis i=10 et Eof devra été positionnée à true.

TEchelle = function(x,y:integer; TypDxy:word; PFx,PFy:PChar):boolean; stdcall;

Cette fonction doit être présente dans la DLL si le message WM_ECHELLE est utilisé par la DLL.
Les paramètres x et y sont les coordonnées en pixels du circuit PElectro. TypDxy est un entier indiquant le type d'échelle utilisée dans les paramètres de retour PFx et PFy. PFx et PFy sont les caractères des coordonnées x,y exprimées dans l'échelle TypDxy qui seront directement affichés dans la fenêtre des contrôles.
La fonction TEchelle pourrait être la suivante. Pixels pour les points à l'écran (0), MIL pour CiDess (1), Pouces (2) pour les circuits gravés (exemple CiDess).

```
begin
  result:=false;
  case TypDxy of
    0: begin
       StrLCopy(PFx,PChar(inttostr(x)),cstMaxChars-1);
       StrLCopy(PFy,PChar(inttostr(y)),cstMaxChars-1);
    end;
    1: begin
       StrLCopy(PFx,PChar(inttostr(round(x/4)*50)),cstMaxChars-1);
       StrLCopy(PFy,PChar(inttostr(round(y/4)*50)),cstMaxChars-1);
    end;
    2: begin
       StrLCopy(PFx,PChar(format('%7.2f',[round(x/4)*0.254/2])),cstMaxChars-1);
       StrLCopy(PFy,PChar(format('%7.2f',[round(y/4)*0.254/2])),cstMaxChars-1);
    end;
    else result:=true;
  end;
end;
```

TGetParam = procedure(ListParam: PChar; n: integer;var H: HBITMAP; NoLg: word); stdcall;

Cette procédure doit être présente dans la DLL si la constante Parameter figure comme opération de la DLL.
ListParam contient le nom qui figure dans la liste de la fenêtre des paramètres (menu Paramètres->Options générales).
n le nombre de caractères +1.
H est l'handle par lequel une seule image doit être transmise. Cette image figurera aussi dans la liste de la fenêtre des paramètres de PElectro.
NoLg contient le numéro de langue (exemple $0C pour le français, voir plus haut).
Exemple:

```
procedure GetParam(List:Pchar; n:integer;var H:HBITMAP; NoLg:word); stdcall;
begin
  StrLCopy(List,PChar(MesLangID(NumLg,13)),n-1);
  H := LoadBitmap(hInstance,'CiDess');
end;
```

TActivParam =function(H: HWND;var TBDll: TabDll; Parm: PTPa):boolean; stdcall;

Cette fonction doit être présente dans la DLL si la constante Parameter figure comme opération de la DLL.
Cette fonction permet à la DLL d'afficher sa fenêtre de paramètres.
H est le handle de PElectro.
TBDll est la liste des DLLs chargées par PElectro.
Parm est la liste des paramètres généraux de l'application.
Le retour de fonction doit être à false si aucune erreur n'a été détectée.

TNumChgCompo=function(H:HWND; i,ACount: word;var A:PTAc;var TBDll: TabDll;var Parm: PTPa; Ctrl: boolean; PagItem,JNum: integer):boolean; stdcall;

Cette routine permet de savoir si la touche contrôle a été pressée sur la liste des composants pour éventuellement un changement de référence. Attention cette fonction n'est active que si la DLL est du type Composant.
La fonction TNumChgCompo doit, elle-même, appeler la fonction TChangeCompo.

Trois paramètres supplémentaires Ctrl, PagItem et JNum permettent de gérer le changement de référence.
Ctrl contient 'true' si la touche de même nom est utilisée.
PagItem est l'indice de la DLL sur laquelle le changement de référence a été demandé.
JNum est le RNumCompo de la nouvelle référence.
i est l'indice de l'ancienne référence.
Un exemple de code est donné ci-dessous (cas des transistors).

```
function NumChgCompo(h:HWND; i,ACount:word;var A:PTAc;
                var TBDll: TabDll; Pa:PTPa; PCtrl:boolean; PagItem,JNum:integer):boolean; stdcall;
var
  j1:word;
  k1: integer;
begin
  result:=true;
  j1 := A[i].RI;
  if (PCtrl) and
     (PagItem<>-1) then begin
     if (A[i].RNumCompo<>JNum) and
       (TBDll[PagItem-1].Composant=A[i].RComposant) then begin
       LoadDB('DBTrans.xml',k1);
       setlength(TabV,k1);
       ReadNumDB(JNum+1,TabV);
       inc(j1);
       if R^[j1].RDonne[9]=TabV[0] then begin
         A^[i].RNumCompo:=JNum;
         R^[j1].RNumCompo := JNum;
         inc(j1,2);
         R^[j1].RNumCompo := JNum;
         inc(j1);
         R^[j1].RNumCompo := JNum;
         inc(j1);
         R^[j1].RNumCompo := JNum;
         inc(j1);
         R^[j1].RNumCompo := JNum;
       end;
       CloseDB;
     end;
  end;
  result:=ChangeCompo(h,i,j,A,R,TBDll,Pa);
end;
```

```
 TCalCompo =  function(Op: word;var A:PTAc;var R:PTRc; NumLg: word;var TBDll: TabDll; i: integer;var
Ai,Vi: TI;out iq: double;var Vm: TI; i1: integer;var Par,Par2: TI; Dxt,Ta,T: double;Def: boolean; Ano: PChar;
n: integer;var TY: TA; out Ok: smallint;var Parm:PTPa; iB:integer):boolean; stdcall;
```

Cette fonction permet de calculer le courant ou la tension d'un sous composant R. Exemple i:=v/r pour une résistance simple.

Op détermine l'opération en cours. Voir exemple de DLLs et le programme oscilloscope sur mon site.
NumLg contient le numéro de langue (exemple $0C pour le français, voir plus haut).
Ok permet en sortie de la fonction d'effectuer une opération particulière. Voir PDF des différents instruments figurant dans mon site.
n taille maximum+1 de la chaîne des Ano. A zéro si Ano n'est pas disponible.
i est le rang du composant dans la table A.
i1 est le rang du sous composant de la table R.
Ai et Vi sont les tables de courant et de tension pour les composants liés (source courant/courant, etc.).
iq est le courant ou la tension calculé par le sous composant R[i1].
Dxt le temps en cours de calcul.
Ta la température ambiante.

T le delta de temps en cours.
iB le nombre de point pour le temps en cours.
Parm la table des paramètres de PElectro.
Ano la liste des anomalies trouvées mises sous forme de chaînes.
TY la table des valeurs min et max que le composant peut normalement prendre.
Vm la table des valeurs (courant ou tension) calculées par linéarisation des dérivées partielles en cours.
Par le paramètre du sous composant R[i1]. Par exemple la tension précédente d'une capacité.
Par2 le paramètre du composant A[i]. Il s'agit de la température précédente du composant.

TInfoCompo = Procedure(H: HWND; i,ACount:word;var A:PTAc;var TBDll: TabDll;var Parm:PTPa; ListInfo:PChar; n:integer; NumLg:word); stdcall

Cette fonction peut être présente dans la DLL si la constante Composant ou Instrument figure comme opération de la DLL.
Cette fonction permet d'afficher dans une infobulle les caractéristiques principales du composant. Si cette procédure n'existe pas, alors c'est la première valeur, si elle n'est pas vide, figurant dans le composant (Adonne[0]) qui est affichée, sinon c'est un message indiquant que les informations ne sont pas disponibles qui s'affiche.

i est le rang du composant dans la table A.
Parm la table des paramètres de PElectro.
ListInfo est une liste en sortie des valeurs d'informations du composant, séparées par un point-virgule ';'.
n est le nombre de caractères maximum+1 de ListInfo.
NumLg contient le numéro de langue (exemple $0C pour le français, voir plus haut).

TInitParCompo=function(H:HWND; i:word;var j: word;var A:PTAc;var R:PTRc;var TBDll:TabDll; var T:word; var Parm: PTPa; NoLg:word ; Tval: PTPa = nil ; Bval: boolean = false):boolean; stdcall;

Cette fonction peut être présente dans la DLL si la constante Composant figure comme opération de la DLL.
Cette fonction permet de mettre à jour les paramètres d'un composant AComposant (elle est utilisée dans les instruments pour initialiser les paramètres voir par exemple les Composants CMOS).

Les paramètres:
H correspond au handle de l'application PElectro.
i est l'indice de l'endroit où est l'AComposant.
j est l'indice à partir de l'endroit où les RComposant doivent être mis à jour.
A est un pointeur sur la table des AComposant.
R est un pointeur sur la table des RComposant.
TBDll est le tableau des informations relatives aux DLLs.
T est le rang en cours pour les sous-composants.
Parm un pointeur sur la table des paramètres.
NoLg est la langue utilisée.
Si la fonction s'est bien terminée, le retour de la fonction doit être à true.

TTypeCompoVisual=function(A: AComposant;out TypeDonVisual:TTypeCompoDonVisual; var Pa: PTPa): byte; stdcall;

Cette fonction retourne TypeDonVisual de type TTypeCompoDonVisual (découpage fourni plus haut) et le type de composant à afficher de 1 à 5 dans une zone 'byte'.

A est la table des AComposant.

TCalOpera = function(var A:PTAc;var R:PTRc; var Vm, Vj: TI ;NLg:word;var TBDll:TabDll; i,n: integer; out Compo: TTabCompoOp; i1: integer; iB: integer; Dxt,Ta,T: double; Ano: Pchar; var Parm:PTPa; var Par,Par2: TI):boolean; stdcall;

Cette fonction permet de retourner Compo (TTabCompoOp) qui est la zone décrite plus haut.
Chaque DLLs de type InstruBox doit avoir cette fonction ainsi que la DLL WH00. Les champs TTabCompoOp sont renseignés en fonction des composants.

A composants du schéma.
R sous-composants.
NLg langue utilisée.
TBDll table des DLLs chargées.
i est le rang du composant dans la table R.
n taille maximum+1 des chaines Ano.
i1 est l'indice des R sans tenir compte des sous-composants masses et en courts-circuits.
iB le nombre de point pour le temps en cours.
Dxt le temps en cours de calcul.
Ta la température ambiante.
T le delta de temps en cours.
Parm la table des paramètres de PElectro.
Par le paramètre du sous composant R[i]. Par exemple la tension précédente d'une capacité.
Par2 le paramètre du composant A[?] (? étant l'indice AffiCompo du sous-composant). Il s'agit de la température précédente du composant.
Le retour, est normalement à False.

TPElectro = procedure(H,Main: HWND; NumLg: word; Var TBDll: TabDll; VersPElec:Pchar); stdcall;

Cette fonction TPElectro qui doit être présente dans la DLL, l'informe sur
H le handle de l'application.
Main le handle de la fenêtre de PElectro.
La langue NumLg utilisée.
TBDll, la table des DLLs chargées.
Et VersPElec le numéro de version de l'application.

TMesPElec = function(H: HWND; Nlg: word;Var TBDll: TabDll; VersPElec: PChar; var Parm: PTPa; var MesIn: TMessage; Fmain: Pointer):boolean; stdcall;

Cette fonction est utilisée uniquement dans la première DLL qui comporte la constante Tools. Elle affiche le message qui figure dans TMessage avec la fenêtre FMain de l'appelant.
H est le handle de l'application.
NLg la langue.
TBDll la table des DLLs.
VersPElec la version de l'application.
Parm les paramètres.
MesIn le message (voir format du message WM_ADDMES).
FMain, la fenêtre TForm de l'application.
Le retour, est normalement à false.

TAltPElec = function(H: HWND; Nlg: word;var TBDll: TabDll; VersPElec: PChar; var Parm: PTPa; AltIn: pointer):boolean; stdcall;

Cette fonction est utilisée uniquement dans la première DLL qui comporte la constante Tools. Elle doit fournir les paramètres utilisés pour l'affichage des messages du composant TJvDesktopAlert utilisé par PElectro. Il faut que la DLL, qui utilise cette fonction, intègre dans sa gestion ce composant.
H est le handle de l'application.
NLg, la langue utilisée.
TBDll, la table des DLLs.
VersPElec, la version de l'application.
Parm est la table des paramètres.
AltIn, un pointer sur l'instance du composant TJvDesktopAlert.
En retour,
AltPElec doit renvoyer true si l'affichage des messages d'alerte est autorisé sinon false.

Pour des exemples, voir les PDF sur mon site.
