Dans mes premiers programmes (Acquis1 1991) , j'ai mis au point les fichiers de configuration.  
L'idée était simplement de sauver des variables globales dans un fichier  et de les récupérer.  
J'ai utilisé Dcfg1.pas (ou son ancêtre).  
  
Le principe est le suivant: je commence par construire un tableau de records  
typeVarConf=record  
                MotCle:typeMotCle;  
                Taille:word;  
                Pvar:pointer;  
                 ...// on oublie la suite pour l'instant  
              end;  
en appelant setvarConf1 pour chaque variable que je veux sauver.  Les paramètres de la fonction sont  
  - un mot clé  
  - la taille de la variable  
  - l'adresse de la variable  
  
Une fois le tableau construit, on appelle EcrireCfg1 pour écrire le fichier  
ou LireCfg1 pour lire le fichier.  
  
Le fichier contient une sucession de sous-blocs  
  -mot-clé de longueur fixe (string[12]) donc 13 octets  
  -taille de la variable (smallint)  sur deux octets       // la taille de la variable ne peut excéder 65536 octets  
  -variable qui occupe taille octets.  
  
Pour utiliser Dcfg1, on écrit:  
  
allouerCfg1();  
setvarConf1('X1', x1, sizeof(x1));  
setvarConf1('X2', x2, sizeof(x2));  
setvarConf1('X3', x3, sizeof(x3));  
....  
ecrireCfg1 ou lireCfg1  
resetCfg1;  
  
Ce système était déjà à la fois simple et très souple. J'ai pu ajouter peu à peu des variables de configuration dans Elphy tout en restant compatible avec les versions antérieures.  
J'utilise toujours Dcfg1 pour écrire (et lire) la première partie des fichiers de configuration Elphy (fichiers .GFC)  
Voir mdac.pas LoadGFC et SaveGFC  
  
En fait, après ces variables de configuration, il y a d'autres infos.  
  
Par la suite, j'ai amélioré le système avec varconf1.pas afin de sauver/charger des objets et plus seulement des variables globales.  
La classe TblocConf va gérer une liste de variables  en créant un tableau de TypeVarConf  
Ces variables seront en général des champs d'un objet.  
  
Dans typeVarConf, on trouve encore  
  Pvar : l'adresse de la variable  
  varSize: la taille de la variable. Au début c'était un smallint (avant 2001), ensuite un longint, d'où l'astuce qui consiste à stocker $FFFF suivi d'un entier long quand on veut dépasser 65536.  
  MotCle: une chaine de longueur quelconque (ou presque)  
  
Donc, sur le disque, pour chaque variable, on trouvera un sous bloc  
- un mot clé de longueur quelconque sous la forme d'une chaine pascal. Le premier octet contient la longueur de la chaine L, les L suivants contiennent la chaine.  
- un entier sur 2octets représentant la taille de la variable. Si cet entier vaut $FFFF, il est suivi par un entier sur 4 octets contenant la vraie taille de la variable.  
- la variable proprement dite.  
Tous ces sous blocs sont rangés à l'intérieur d'un bloc Elphy qui a la structure  
BlocSize (4octets)   // taille totale du bloc Elphy  
BlocName( pascal string)  // occupe length(BlocName) +1 octets sur le disque  
Infos (tous les sous-blocs+ éventuellement d'autres infos)  
  
La méthode pour lire/écrire un bloc  devient:  
Create;  
setvarConf('X1', x1, sizeof(x1));  
setvarConf('X2', x2, sizeof(x2));  
setvarConf('X3', x3, sizeof(x3));  
....  
ecrire ou lire;  
free;  
  
Bien entendu, il y a des perfectionnements avec  
setStringConf pour les chaines dynamiques  
setPropConf pour les propriétés  
setDynConf pour les variables dynamiques (qui nécessitent une allocation mémoire avant chargement)  
  
Pour un lire/Ecrire un objet Elphy, on utilise  saveToStream et loadFromStream (voir typeUO dans stmObj.pas)  
Quand ces méthodes ne sont pas surchargées on utilise  
  
procedure typeUO.saveToStreamBase(f:TStream;Fdata:boolean);  
var  
  conf:TblocConf;  
begin  
  conf:=TblocConf.create(stmClassName);  
  BuildInfo(conf,false,true);  
  conf.ecrire(f);  
  conf.free;  
  completeSaveInfo;  
  
  if Fdata then saveData(f);  
end;  
  
et  
  
function typeUO.loadFromStreamBase(f:Tstream;size:LongWord;Fdata:boolean):boolean;  
var  
  ok:boolean;  
  conf:TblocConf;  
  st:AnsiString;  
  posf:int64;  
begin  
  conf:=TblocConf.create(stmClassName);  
  BuildInfo(conf,true,true);  
  ok:=(conf.lire1(f,size)=0);  
  conf.free;  
  
  .....  
end;  
  
BuildInfo contient simplement une succession d'instructions setvarConf  
Donc, pour les objets les plus simples, il suffit de surcharger BuildInfo pour que l'objet soit sauvé/chargé correctement.   
On voit ci-dessus que le nom du bloc est stmClassName, une propriété associée à chaque classe descendant de typeUO (ex: TVector, TVSbitmap...)  
  
 Souvent SaveData ne fait rien. Sinon, saveData écrit un bloc indépendant à la suite du bloc stmClassName   
CompleteSaveInfo et CompleteLoadInfo ne font rien par défaut. Elles peuvent contenir des actions à réaliser après la lecture/écriture.