Présentation : Delphi 2009 et l'UNICODE

par Damien Griessinger (HpAlpha)

Date de publication : 04 septembre 2008

Dernière mise à jour :

Delphi est, comme vous le savez, un outil de programmation et un langage très performant, Borland puis CodeGear et maintenant Embarcadero ont, au fils des versions, essayé de nous délivrer un produit de plus en plus complet, cependant il y avait encore du retard comme la mobilité et l'internationalisation.
La dernière mouture a clairement axé sa cible sur l'international, un support complet de l'unicode, c'est LA grande nouveauté.
Le développement pour mobiles n'étant pas encore prévu pour cette version, c'est bien dommage.
Au cours de cet article, je vais vous présenter comment Delphi a intégré l'unicode et aussi quelles en sont les conséquences.
Bonne lecture.

I - L'unicode ? Qu'est-ce donc ?

L'unicode est une norme qui permet de représenter n'importe quel caractère de n'importe quel langue en un identifiant numérique.
Ca ne vous rappelle rien ? Espace = 32, A = 65, B = 66, ... La table ASCII ! enfin, oui et non, si vous vous souvenez bien, la table contient 255 caractères ... et c'est là un problème dès que l'on sort de notre pays !
Comment faire tenir dans cette table l'alphabet francais, cyrillique, chinois, japonais ou encore hébreu ??? C'est tout simplement impossible.
Il y avait une solution : les pages codes, chaque lot de caractères était dans une page code différente (pour la france 850, ...). Ca marchait très bien sauf que ... une application conçu au Japon avait un rendu inattendu en France !
Pire, on ne pouvait pas avoir dans le même écran du Russe et de l'hébreu (certe c'est un exemple un peu extrême), bref un casse-tête dès que l'on commence à créer des applications pour l'international.
C'est là que l'Unicode est intéressant !
La table ASCII est codé sur 1 octet (soit 8 bits) donc 256 possibilités, donc il suffit d'augmenter le nombre d'octets.
C'est la naissance de l'Universal Transformation Format : UTF
il existe 3 UTF :
* UTF-8
* UTF-16
* UTF-32

Je passerai les détails car le fonctionnement dépasserait le cadre de cet article, mais il faut savoir que l'UTF est codé sur 1, 2 ou 4 octets, l'UTF-8 est par exemple codé entre 1 et 4 octets (il est dit variable)
pour le développeur c'est une vraie problématique. En effet, notre cher Delphi travaille avec des string et des char ... sur 1 octet, c'est pour cela que Delphi 2009 est très attendu, il gère de manière transparente l'Unicode.

II - Avant Delphi 2009 ?

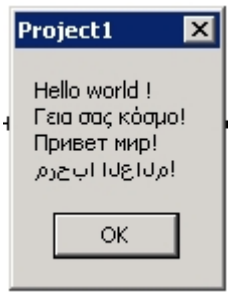
L'unicode existe depuis longtemps. Il est même un peu faux de prétendre que les versions antérieures de Delphi ne gère pas l'unicode, son exploitation se passait au travers du type **widestring**
Bien que pour le code on pouvait utiliser le type widestring, on ne pouvait cependant pas afficher des caractères unicodes dans les VCL, il y a eu quelques composants comme les **TMS Unicode** (ex TNT Control) qui le permettait, et une tentative plutôt originale d'interception des appels API avec **UTF8VCL**, cette unité permet d'utiliser les VCL de base, et en temps réel, lors de l'exécution de votre programme, change les appels API par des appels unicodes, malheureusement, le projet n'est plus mis à jour et souffre de fuite mémoire.

III - IDE

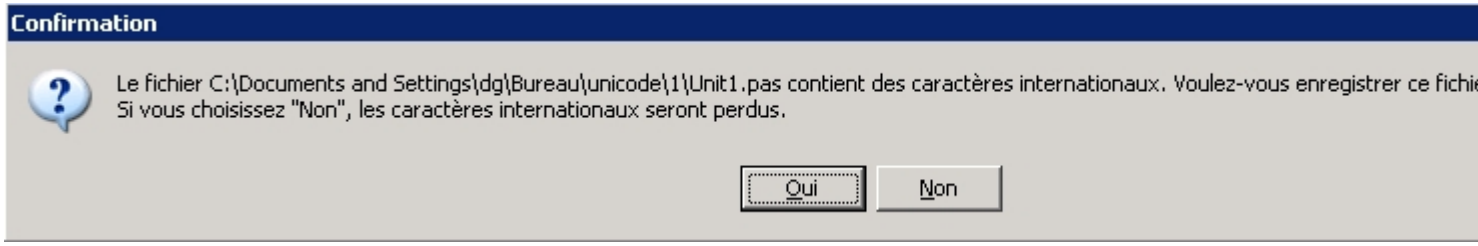
Avant de parler des changements dans le langage, sachez que l'interface de Delphi aussi a eu le droit à son "unicodification" (excusez moi de ce néologisme, mais ce serait la définition la plus adéquate).
La saisie de caractères ne pose pas de problème dans le code source, exemple :

```
var grec,arabe,russe,anglais : string;
begin
  grec:='Γεια σας κόσμο!';
  arabe:='مرحبا بالعالم!';
  russe:='Привет мир!';
  anglais:='Hello world !';
 showmessage(anglais+slinebreak+grec+slinebreak+russe+slinebreak+arabe);
end;
```

Nous donne bien :



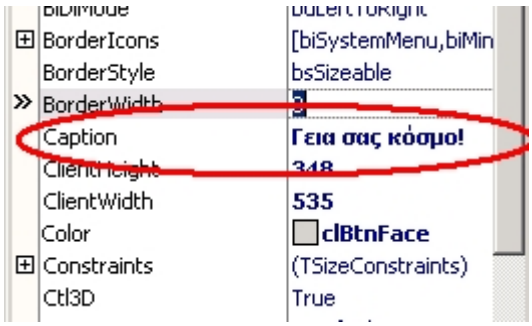
A noter que lorque l'on sauve le projet, une alerte nous propose d'enregistrer le fichier au format UTF-8 :



Certes, on peut entrer des chaines en unicode, mais on peut aussi nommer les fonctions, variables ou encore les classes avec n'importe quel caractère :

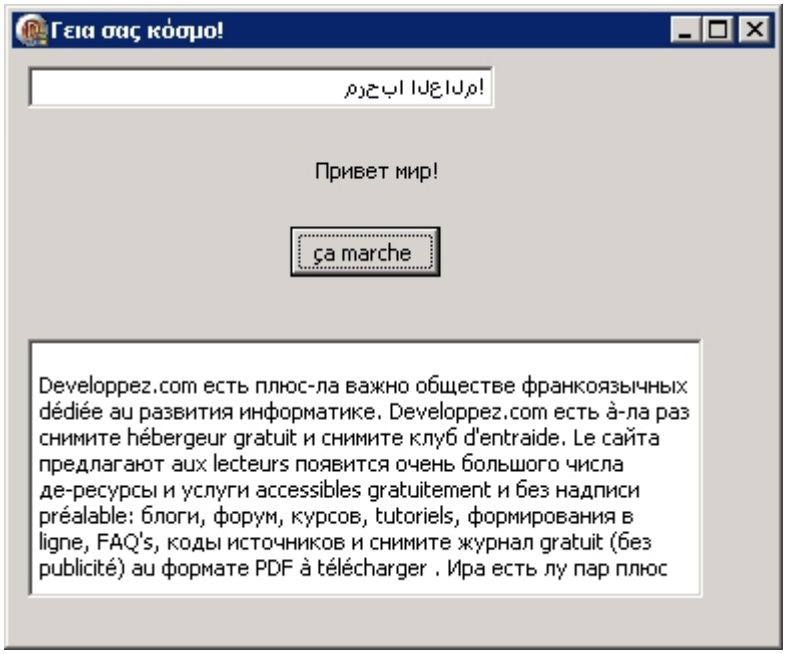
```
type Tmaclasseàmoi = class
  école : boolean;
  hôte : string;
end;

function cache_moi_ça(ici_et_là:integer):integer;
var   αναγνωριστικό : integer;
begin
  αναγνωριστικό:=ici_et_là*333;
  result:=αναγνωριστικό;
end;
```

Bien entendu, ce qui est valable pour l'editeur de code, l'est aussi pour la fiche et l'inspecteur d'objet :



IV - La VCL

La VCL a complètement été retravaillée pour être compatible avec l'unicode, donc à la corbeille TNTControl et compagnie, au revoir l'instable et buggée utf8vcl...
Les TEdit, TMemo, TLabel sont désormais prêts pour l'international et ce, sans modification, un bon point.



V - Le langage

Le langage a subi de profondes modifications comme par exemple le Char est devenu WideChar et String en WideString.
Le bout de code suivant montre bien que l'unicode est par défaut :

```
procedure test;
var
  CharBuffer: array[0..100] of Char;
  AnsiCharBuffer: array[0..100] of AnsiChar;
  WideCharBuffer: array[0..100] of WideChar;
begin
 showmessage(format('array[0..100] of Char : length = %d  sizeof = %d',
[length(charbuffer),sizeof(charBuffer)]));
 showmessage(format('array[0..100] of AnsiChar : length = %d  sizeof = %d',
[length(AnsiCharBuffer),sizeof(AnsiCharBuffer)]));
 showmessage(format('array[0..100] of WideChar : length = %d  sizeof = %d',
[length(WideCharBuffer),sizeof(WideCharBuffer)]));

 {
   ca retourne :
     array[0..100] of Char : length = 101  sizeof = 202
     array[0..100] of AnsiChar : length = 101  sizeof = 101
     array[0..100] of WideChar : length = 101  sizeof = 202
 }
end;
```

Il y a l'apparition un nouveau type : UnicodeString, utile pour clarifier le code
Dans l'unité system.pas, il y a du changement, notamment s'il l'on compare avec la version précédente :
Delphi 2007 : UTF8String = type string;
Delphi 2009 : UTF8String = type AnsiString(65001);
AnsiString(65001) !?! On peut faire ça ? Et bien oui, 65001 est la page code de l'unicode dans windows.
En fait, le compilateur se chargera tout seul de la conversion entre la chaine d'entrée et de sortie.
A noter qu'il existe aussi **RawByteString = type AnsiString($ffff);** la chaine ne subira pas de conversion.
Une nouvelle unité fait son entrée : character.pas, elle contient, entre autre, une classe TCharacter, et plein de fonctions comme **IsDigit(C: Char): Boolean; IsNumber(const S: string; Index: Integer): Boolean;** , ... (je vous laisse deviner à quoi elles servent) Cette unité est spécialement conçue pour gérer correctement l'unicode.
Les classes comme TStringList ont subi un petit lifting, ainsi : **MaStringList.SaveToFile('lisezmoi.txt');**

devient, si on veut forcer le codage : **MaStringList.SaveToFile('lisezmoi.txt', TEncoding.Unicode);**

VI - La compatibilité avec l'existant ?

C'est là où le bâs blesse. Certes si votre application est 100% VCL native et que vous utilisez normalement les chaines, votre application se compilera sans problème. Mais si vous utilisez les chaines comme buffer par exemple ou des composants additionnels, il faudra adapter :

- Vous avez les codes source: il faut mettre les mains dans le cambouis
- Vous n'avez pas les codes source: if faut attendre une mise à jour du composant

Point positif, les warnings sont très explicites, ce qui facilite la conversion. il faut par conséquent utiliser AnsiChar et AnsiString pour retourner aux types précédents.
Les fonctions classiques sont conservées, comme Length, Copy ou autre Delete. Elles sont parfaitement unicode, c'est à dire que Length retournera le bon nombre de caractères **(length != sizeof)**.
Vous pouvez aussi commencer à préparer vos applications existantes pour le passage à l'unicode, par exemple, la directive UNICODE n'existe pas dans les anciennes versions, on peut très bien imaginer faire :

```
{$IFDEF UNICODE}
 type
      machaine : unicodeString;
{$ELSE}
 type
      machaine : AnsiString;
{$ENDIF}
```

L'indexation d'une chaine est exactement la même. Ainsi, quand on fait :

```
var s:unicodestring;
begin
 s:='Hello';
 showmessage(s[2]); // ca affiche bien "e"
end;
```

Il faudra, notament dans le cas d'appel à des API, faire des modifications (exemple venant d'Embarcadero) :

```
var
  Count: Integer;
  Buffer: array[0..MAX_PATH - 1] of Char;
begin
  // Code existant - invalide quand string = UnicodeString
  Count := SizeOf(Buffer);
  GetWindowText(Handle, Buffer, Count);

  // Correct pour l'Unicode
  Count := Length(Buffer);
  GetWindowText(Handle, Buffer, Count);
end;
```

VII - Conclusion

Delphi 2009 est un bon cru pour les amateurs d'unicode, mais c'est aussi un pari risqué pour Embarcadero que de toucher de façon radicale au langage !
Les entreprises ne vont pas pouvoir basculer aussi rapidement d'un projet version delphi 2006/2007 à un Delphi 2009 que si l'on était passait de Delphi 7 à Delphi 2007.

Le "forçage" par défaut de l'unicode est assez génant (mais c'est un passage obligatoire), Il a eu énormement de travail pour rendre compatible par exemple la JVCL, mais, étant un projet open source, des milliers de développeurs ont pu réaliser les mises à jour assez rapidement.
Quid des composants/librairies dont les sociétés n'existent plus ? L'avenir nous le dira.