

# Accentuation correcte en Ada sur Mac

Cet article est certainement valable aussi pour de nombreux langages de programmation. Je vais, cependant, l'illustrer avec le langage Ada.

D'une question apparemment simple "Comment afficher correctement les accents ?" : attention à ne pas être piégé par les apparences.

## 1) Le constat

Prenons l'exemple suivant avec une configuration du Terminal avec le codage UTF-8 et un code source encodé en UTF-8 (par défaut sur Mac) : (Dans les préférences du Terminal, cliquer sur Profils puis sélectionner l'onglet Avancé et dans la rubrique International sélectionner l'encodage UTF-8)

```
$ cat str_UTF-8.adb
with Ada.Text_IO;
use Ada.Text_IO;
procedure Str_UTF-8 is
S : String := "Chaîne accentuée en UTF-8 !!!";
begin
Put_Line(S);
end;
$ gnatmake str_UTF-8.adb
gcc -c str_UTF-8.adb
gnatbind -x str_UTF-8.ali
gnatlink str_UTF-8.ali
$ ./str_UTF-8
Chaîne accentuée en UTF-8 !!!
```

Cela fonctionne très bien en apparence ! Pour la simple raison que la chaîne de caractères n'est qu'une suite d'octets qui sont envoyés directement vers l'affichage, le codage du code source n'est pas altéré.

Par contre, modifions notre programme en n'y ajoutant quelques lignes :

```
$ cat str_UTF-8.adb
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Characters.Handling;
use Ada.Characters.Handling;
procedure Str_UTF-8 is
S : String := "Chaîne accentuée en UTF-8 !!!";
begin
Put_Line(To_Upper(S));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & S'Length'img);
end;
$ gnatmake str_UTF-8.adb
gcc -c str_UTF-8.adb
gnatbind -x str_UTF-8.ali
gnatlink str_UTF-8.ali
$ ./str_UTF-8
CHAÎNE ACCENTUÉE EN UTF8 !!!
123456789012345678901234567890123
Longueur :  30
```

Le nombre de caractères affichés (28) n'est pas cohérent avec la longueur calculée (30). De plus les utilitaires de transformation de caractères ne fonctionnent pas correctement, ici `To_Upper` pour passer en majuscule ne donne pas les bonnes lettres majuscules. Pour la simple raison que l'encodage attendu par Ada n'est pas celui fourni par le code source.

Changeons la configuration de notre terminal pour Latin-1 et prenons un code source encodé en Latin-1 (par défaut sur Windows) :  
(Dans les préférences du Terminal, cliquer sur Profils puis sélectionner l'onglet Avancé et dans la rubrique International sélectionner l'encodage ISO Latin-1)

```
$ cat str_lat1.adb
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Characters.Handling;
use Ada.Characters.Handling;
procedure Str_Lat1 is
S : String := "Chaîne accentuée en Latin-1 !!!";
begin
Put_Line(To_Upper(S));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & S'Length'img);
end;
$ gnatmake str_lat1.adb
gcc -c str_lat1.adb
gnatbind -x str_lat1.ali
gnatlink str_lat1.ali
$ ./str_lat1
CHAÎNE ACCENTUÉE EN LATIN-1 !!!
123456789012345678901234567890123
Longueur : 31
```

Le nombre de caractères affichés (31) est maintenant cohérent avec la longueur calculée (31). De plus les utilitaires de transformation de caractères (ici To\_Upper pour passer en majuscule) fonctionnent également. L'encodage attendu par Ada, Latin-1, est celui fourni par le code source.

## 2) L'explication complète

Ada normalise trois types de représentation d'un caractère : `Character`, `Wide_Character` et `Wide_Wide_Character`. Le codage interne de ces types est plus clairement explicité dans la norme, ce référer au manuel de référence Ada "A.1 The Package Standard" :

"The declaration of type `Character` is based on the standard ISO 8859-1 character set."

"The declaration of type `Wide_Character` is based on the standard ISO/IEC 10646:2011 BMP character set. The first 256 positions have the same contents as type `Character`."

"The declaration of type `Wide_Wide_Character` is based on the full ISO/IEC 10646:2011 character set. The first 65536 positions have the same contents as type `Wide_Character`."

Le type `Character` est donc basé sur le standard ISO 8859-1. La moitié inférieure est identique au codage ASCII standard, mais la moitié supérieure est utilisée pour représenter des caractères complémentaires. Ceux-ci incluent des lettres étendues utilisées par des langues européennes, comme des accents français, les voyelles avec des trémas utilisés en allemand et les lettres supplémentaires utilisées en suédois. Pour une liste complète des codes et leurs codages voir le fichier source de l'unité de bibliothèque `Ada.Characters.Latin_1`.

`Wide_Character` est lui basé sur le standard ISO 10646. Les 256 premières positions ont le même contenu que le type `Character`. Le reste des codes est utilisé pour les langues avec des scripts tels que le Chinois.

`Wide_Wide_Character` permet d'étendre encore plus les possibilités avec des caractères graphiques comme les émoticônes.

## 3) En pratique

La situation dépend du compilateur pour le décodage des sources et les entrées sorties et du système pour l'encodage des sources et l'affichage.

Concernant les sources, le besoin est de récupérer les constantes chaînes de caractères avec le bon codage c'est à dire Latin-1. Le compilateur doit donc convertir le format utilisé en Latin-1.

Avec GNAT, les formats possibles sont les suivants : ASCII et plus largement Latin-1 sans option spécifique. Il existe plusieurs formats "étendus" disponibles pour les systèmes hors Windows. Mais le seul exploitable pour Mac va être UTF-8 avec l'option `-gnatW8`.

Reprenons notre exemple en UTF-8 et notre code source en UTF-8 :  
(Dans les préférences du Terminal, cliquer sur Profils puis sélectionner l'onglet Avancé et dans la rubrique International sélectionner l'encodage UTF-8)

```
$ cat str_UTF-8.adb
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Characters.Handling;
use Ada.Characters.Handling;
procedure Str_UTF-8 is
S : String := "Chaîne accentuée en UTF-8 !!!";
begin
Put_Line(To_Upper(S));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & S'Length'img);
end;
$ gnatmake -gnatW8 str_UTF-8.adb
gcc -c -gnatW8 str_UTF-8.adb
gnatbind -x str_UTF-8.ali
gnatlink str_UTF-8.ali
$ ./str_UTF-8
CHAÎNE ACCENTUÉE EN UTF8 !!!
123456789012345678901234567890123
Longueur : 28
```

La longueur est bonne ainsi que la transformation en majuscules.

Concernant les entrées / sorties le format par défaut de GNAT est le format interne ISO 8859-1 avec l'unité `Ada.Text_IO` et le format du code hexadécimal entre crochets avec l'unité `Ada.Wide_Text_IO`.

Ce format peut être modifié dynamiquement avec le paramètre "FORM" uniquement avec les procédures de l'unité `Wide_Text_IO`. `Text_IO` en bénéficie avec GNAT GPL 2015. Par exemple, une requête `Open` avec le paramètre `FORM` avec la valeur "WCEM=8" permet de lire ou d'écrire du texte au codage UTF-8.

Pour les entrées / sorties standards (Terminal) le codage par défaut est donné à la compilation du programme principal, par exemple, avec l'option `-gnatW8`, ici UTF-8.

Mais que se passe-t-il si le caractère encodé en UTF-8 n'a pas de correspondance en Latin-1 ? C'est ce que nous allons voir en utilisant le type `Wide_Character`.

#### 4) Le type `Wide_Character`

Dupliquons notre programme source encodé en UTF-8 pour ajouter le caractère  $\pi$  qui n'existe pas en Latin-1 :

```
cmd$ cat wstr_utf8.adb
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Characters.Handling;
use Ada.Characters.Handling;
procedure WStr_UTF8 is
S : String := "Périmètre / Diamètre =  $\pi$ ";
begin
Put_Line(To_Upper(S));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & S'Length'img);
end;
cmd$ gnatmake -s -gnatW8 -gnatv wstr_utf8
gcc -c -gnatW8 -gnatv wstr_utf8.adb
```

GNAT GPL 2015 (20150428-49)  
Copyright 1992-2015, Free Software Foundation, Inc.

```
Compiling: wstr_utf8.adb
Source file time stamp: 2016-03-28 10:15:57
Compiled at: 2016-03-28 12:15:58
```

```
6. S : String := "Périmètre / Diamètre =  $\pi$ ";
                                     |
```

```
>>> literal out of range of type Standard.Character
```

```
11 lines: 1 error
gnatmake: "wstr_utf8.adb" compilation error
```

Nous obtenons une erreur claire et cohérente avec ce qui précède.  
Nous devons alors passer à la dimension supérieure en utilisant le type `Wide_Character` :

```
cmd$ cat wstr_utf8.adb
with Ada.Wide_Text_IO;
use Ada.Wide_Text_IO;
with Ada.Wide_Characters.Handling;
use Ada.Wide_Characters.Handling;
```

```

with Ada.Characters.Conversions;
use Ada.Characters.Conversions;
procedure WStr_UTF8 is
S : Wide_String := "Périmètre / Diamètre =  $\pi$ ";
begin
Put_Line(To_Upper(S));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & To_Wide_String(S'Length'img));
end;
cmd$ gnatmake -s -gnatW8 wstr_utf8
gcc -c -gnatW8 wstr_utf8.adb
gnatbind -x wstr_utf8.ali
gnatlink wstr_utf8.ali
cmd$ ./wstr_utf8
PÉRIMÈTRE / DIAMÈTRE =  $\Pi$ 
123456789012345678901234567890123
Longueur : 24

```

Le type `Wide_Character` accepte beaucoup plus de variantes de caractères que le type `Character`. Cependant, il peut s'avérer limité il faudra alors passé au type `Wide_Wide_Character`.

Le choix de du bon type dépendra de l'étendu des caractères à prendre en compte. Pour le français, langue latine, le type `Character` (Latin-1) convient pour la plupart des textes quotidiens. Pour les autres graphies comme le grec voire le braille il faudra passer au `Wide_Character`. Pour les émoticônes, ce sera carrément le `Wide_Wide_Character`.

## 5) La saisie de caractères accentués

Nous avons vu que nous pouvions donc afficher des caractères accentués en prenant quelques précautions, quand est-il de la saisie ?

Reprenons notre programme d'affichage des chaînes de caractères `String` et ajoutons la saisie depuis le terminal, compilons et exécutons le programme en tapant directement dans le terminal les caractères "Saisie accentuée !!!" :

```

cmd$ cat str_get_utf8.adb
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Characters.Handling;
use Ada.Characters.Handling;
procedure Str_Get_UTF8 is
  S : constant String := "Chaîne accentuée en UTF8 !!!";

```

```

    G : constant String := Get_Line;
begin
Put_Line(To_Upper(S));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & S'Length'img);
Put_Line(To_Upper(G));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & G'Length'img);
end;
cmd$ gnatmake -gnatW8 str_get_utf8.adb
gcc -c -gnatW8 str_get_utf8.adb
gnatbind -x str_get_utf8.ali
gnatlink str_get_utf8.ali
cmd$ ./str_get_utf8
Saisie accentuée !!!
CHAÎNE ACCENTUÉE EN UTF8 !!!
123456789012345678901234567890123
Longueur : 28
SAISIE ACCENTUÃ©E !!!
123456789012345678901234567890123
Longueur : 21

```

À l'évidence la saisie ne fonctionne pas de la même façon que l'affichage. En consultant le manuel GNAT à propos de l'option `-gnatW8`, nous trouvons : "Note that the wide character representation that is specified (explicitly or by default) for the main program also acts as the default encoding used for `Wide_Text_IO` files if not specifically overridden by a `WCEM` form parameter."

Ce paragraphe indique que l'option fonctionnera avec la bibliothèque `Wide_Text_IO`. Bien, mais pourquoi avec `Text_IO` l'affichage (`Put_Line`) fonctionne mais pas la saisie (`Get_Line`). L'explication réside dans le sens de conversion. Pour l'affichage, il y a une conversion de `String` (Latin-1) vers UTF8, ce qui est toujours possible. Pour la saisie, il y a une conversion de UTF8 vers `String` (Latin-1), ce qui n'est pas toujours possible. Le choix de `Text_IO.Get_Line` est alors de ne pas faire de conversion. Il faut alors utiliser `Wide_Text_IO.Get_Line` qui rend la conversion possible.



Reprenons notre programme d'affichage des chaînes de caractères Wide\_String et ajoutons la saisie depuis le terminal, compilons et exécutons le programme en tapant directement dans le terminal les caractères "Saisie accentuée !!!" :

```
cmd$ cat wstr_get_utf8.adb
with Ada.Wide_Text_IO;
use Ada.Wide_Text_IO;
with Ada.Wide_Characters.Handling;
use Ada.Wide_Characters.Handling;
with Ada.Characters.Conversions;
use Ada.Characters.Conversions;
procedure WStr_Get_UTF8 is
S : constant Wide_String := "Périmètre / Diamètre =  $\pi$ ";
G : constant Wide_String := Get_Line;
begin
Put_Line(To_Upper(S));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & To_Wide_String(S'Length'img));
Put_Line(To_Upper(G));
Put_Line("123456789012345678901234567890123");
Put_Line("Longueur : " & To_Wide_String(G'Length'img));
end;
cmd$ gnatmake -gnatW8 wstr_get_utf8.adb
gcc -c -gnatW8 wstr_get_utf8.adb
gnatbind -x wstr_get_utf8.ali
gnatlink wstr_get_utf8.ali
cmd$ ./wstr_get_utf8
Saisie accentuée !!!
PÉRIMÈTRE / DIAMÈTRE =  $\pi$ 
123456789012345678901234567890123
Longueur : 24
SAISIE ACCENTUÉE !!!
123456789012345678901234567890123
Longueur : 20
```

Cela fonctionne maintenant comme attendu.

Pascal Pignard, mars 2005, mars 2016, avril 2017.