Utilisation des fichiers typés sous Delphi 8

par Bruno Guérangé (nono40.developpez.com)

Date de publication : 14/02/2004

Dernière mise à jour : 01/07/2004

Ce document présente une méthode d'utilisation des fichiers typés dans Delphi 8 depuis la suppression du type file dans Delphi 8.

Introduction

Delphi 8 ne permet plus l'utilisation des fichiers typés. Ceci car l'affectation des variables ne peut plus être effectuée par bloc d'octets ce qui était le cas avec les fonctions Read ou Write.

La méthode présentée ici donne un moyen de contourner le problème et d'utiliser des fichiers disques issus de fichiers typés des versions antérieures de Delphi.

Ceci n'est présenté qu'a titre de compatibilité pour la migration d'application Delphi 7, ou le besoin indispensable de lire des fichiers écrits par des versions antérieures de Delphi.

I - Classe TFileRecord

I-A - Pourquoi créer cette nouvelle classe ?

Toutes les procédures de gestion des fichiers typés (Rewrite, Reset, Seek, etc..) ne sont plus utilisables dans Delphi 8. La classe TFileRecord va permettre de créer de nouveau ces procédures en tant que méthode d'une classe. Le but est de pouvoir retranscrire le code existant Delphi 7 en un minimum de temps.

I-B - Description de la classe TFileRecord

La déclaration de la classe TFileRecord est la suivante :

```
Type
  TFileRecord=Class Abstract(TObject)
  Private
    FNom    :String;
    FStream :TFileStream;
    FTaille :Integer;
  Protected
    Property  Stream:TFileStream Read FStream;
    Procedure ReadShortString(Var Chaine:ShortString;Longueur:Integer);
    Procedure WriteShortString(Const Chaine:ShortString;Longueur:Integer);
  Public
    Constructor Create(Taille:Integer);Override;
    Destructor Destroy;Override;
    Procedure AssignFile(Const NomFichier:String);
    Procedure Reset;
    Procedure ReWrite;
    Procedure Seek(N:Integer);
    Function  EOF:Boolean;
    Function  FileSize:Integer;
    Procedure CloseFile;
  End;
```

Comme vous pouvez le voir, cette classe est abstraite et ne peut donc pas être instanciée. Elle présente toutes les méthodes de base utiles ensuite pour la gestion des fichiers typés.

Cette classe est prévue afin de générer une classe descendante par format de fichier typé. Le chapitre suivant va montrer la méthode utilisée pour adpater cette classe à un type Record connu.

Les méthodes protégée pour la lecture des chaines courtes, sont ici pour assurer la lecture dans un flux d'une chaine de type ShortString. Les chaines longues ne pouvants pas être incluses dans les fichiers typés, il est donc fréquent d'y trouver des chaines courtes.

Les méthodes d'accès au fichier (Rewrite, Reset, AssignFile, etc) vont permettre de garder la structure du programme de lecture/écriture du fichier typé.

Cette classe n'existe pas dans Delphi 8 et a été créée pour les besoins de cet article. Le code source de cette classe est disponible dans l'unité UFileRecord.pas située dans le **zip de l'exemple**.

II - Accéder au fichier typé

Ce paragraphe va présenter l'accès à un fichier typé dont les éléments sont du type suivant :

```
Type
  TElement=Packed Record
    eleEntier : Integer;
    eleReel   : Double;
    eleChaine : String[100];
  End;
```

II-A - Création de la classe TFileElement

Pour accéder au fichier contenant des TElements, nous allons créer une classe descendante de TFileRecord nommée TFileElement. L'unité UFileRecord n'a pas besoin d'être modifiée et peut donc être ajoutée telle quelle à votre projet.

Cette classe doit ressembler à la déclaration suivante :

```
Type
  TFileElement=Class(TFileRecord)
  Public
    Constructor Create;
    Procedure Read(Var Element:TElement);
    Procedure Write(Const Element:TElement);
  End;
```

La surcharge du constructeur est obligatoire afin de préciser la taille d'un élément du fichier. les deux autres méthodes sont ici pour la lecture et l'écriture des enregistrements.

II-B - Surcharge du constructeur

La surcharge du constructeur sera la suivante dans le cas ou la taille de l'enregistrement est connue :

```
constructor TFileElement.Create;
begin
  inherited;
  Taille:=113;
end;
```

A noter que la taille de l'élément doit être spécifiée directement. La fonction SizeOf ne retourne plus la valeur correcte sous Delphi 8. La valeur peut être obtenue par l'utilisation de SizeOf sous Delphi 7.

Une autre solution est de créer un fichier temporaire d'un élément dans le constructeur et d'en obtenir la taille. Cette solution sera préférable dans le cas ou la taille n'est pas connue, ou que la structure du fichier sera amenée à être modifiée par la suite.

```
constructor TFileElement.Create;
Var Element:TElement;
begin
  inherited;
  AssignFile(ExtractFilePath(Application.ExeName)+'~temp.dat');
  Rewrite;
  Write(Element);
  Taille:=Stream.Size;
  CloseFile;
  AssignFile('');
  DeleteFile(ExtractFilePath(Application.ExeName)+'~temp.dat');
end;
```

Remarque, dans le cas ou vous n'utilisez pas les méthode TFileRecord.FileSize ou TFileRecord.Seek (avec une valeur non nulle), la taille de l'enregistrement n'est pas utile. Dans ce cas elle peut être fixée à 1 quelle que soit la taille réelle de l'enregistrement.

II-C - Méthodes de lecture et d'écriture d'un élément

Il faut ensuite construire les méthodes de lecture et d'écriture d'un élément. Comme la lecture globale n'est plus autorisée, les champs seront lus un par un dans l'ordre du Record.

Dans notre exemple les méthode de lecture et d'écriture seront les suivantes :

```
procedure TFileElement.Read(var Element: TElement);
begin
  If Assigned(Stream) Then
  Begin
    Stream.Read(Element.eleEntier);
    Stream.Read(Element.eleReel);
    ReadShortString(Element.eleChaine,High(Element.eleChaine));
  End;
end;

procedure TFileElement.Write(const Element: TElement);
begin
  If Assigned(Stream) Then
  Begin
    Stream.Write(Element.eleEntier);
    Stream.Write(Element.eleReel);
    WriteShortString(Element.eleChaine,High(Element.eleChaine));
  End;
end;
```

ATTENTION : dans l'exemple donné ici le Record est de type Packed, dans le cas contraire, il peut y avoir des octets inutilisés d'alignement entre les champs. Il faut donc lire et écrire ces octets intermédiaires.

Le code suivant donne un exemple d'ajout de la lecture d'un octet d'alignement dans le flux. Il n'y a pas malheureusement de règle simple pour connaitre où sont les octets d'alignement car Delphi 1, Delphi 2, Delphi 3/4, Delphi 5/6/7 donnais des résultats différents sur chaque version.

```
procedure TFileUnAutreElement.Read(var UnAutreElement: TUnAutreElement);
Var Octet:Byte;
begin
  If Assigned(Stream) Then
  Begin
    Stream.Read(UnAutreElement.eleEntier);
    Stream.Read(UnAutreElement.eleReel);
    Stream.Read(octet); // Lecture d'un octet d'alignement.
    ReadShortString(eleChaine,High(UnAutreElement.eleChaine));
  End;
end;
```

II-D - Exemple de lecture séquentielle d'un fichier

Sous Delphi 7 la méthode suivante de parcours complet d'un fichier pour remplir un tableau ou une autre structure est fréquente :

code Delphi 7

```
procedure TForm1.btnLireClick(Sender: TObject);
Var Fichier:File Of TElement;
    Element:TElement;
begin
  AssignFile(Fichier, ExtractFilePath(Application.ExeName)+'fichier.dat');
```

code Delphi 7
```
   Reset(Fichier);
   Try
     Liste.Clear;
     While Not Eof(Fichier) Do
     Begin
       Read(Fichier,Element);
       With Liste.Items.Add Do
       Begin
         Caption := Element.eleChaine;
         SubItems.Add(IntToStr(Element.eleEntier));
         SubItems.Add(Format('%1.3f',[Element.eleReel]));
       End;
     End;
   Finally
     CloseFile(Fichier);
   end;
 end;
```

La classe TFileElement va permettre de conserver la structure du code en créant une instance de la classe. Après la création de la classe, les anciens appels de procédures seront simplement remplacés par les appels des méthodes de même nom de la classe.

code Delphi 8
```
 procedure TForm1.btnLireClick(Sender: TObject);
 Var Fichier:TFileElement;
     Element:TElement;
 begin
   Fichier:=TFileElement.Create;
   Try
     Fichier.AssignFile(ExtractFilePath(Application.ExeName)+'fichier.dat');
     Fichier.Reset;
     Liste.Clear;
     While Not Fichier.EOF Do
     Begin
       Fichier.Read(Element);
       With Liste.Items.Add Do
       Begin
         Caption := Element.eleChaine;
         SubItems.Add(IntToStr(Element.eleEntier));
         SubItems.Add(Format('%1.3f',[Element.eleReel]));
       End;
     End;
   Finally
     Fichier.CloseFile;
     Fichier.Free;
   end;
 end;
```

On voit dans l'exemple ci-dessus que la migration du code est très rapide dés que la classe est créée de façon correcte.

Le code d'écriture sequentiel du même fichier est par exemple sous Delphi 7 :

code Delphi 7
```
 procedure TForm1.btnSauverClick(Sender: TObject);
 Var Fichier:File Of TElement;
     i       :Integer;
     Element:TElement;
 begin
   AssignFile(Fichier, ExtractFilePath(Application.ExeName)+'fichier.dat');
   ReWrite(Fichier);
   Try
     For i:=0 To Liste.Items.Count-1 Do
     Begin
       With Liste.Items[i] Do
```

code Delphi 7
```
      Begin
        Element.eleEntier :=StrToIntDef   (SubItems[0],0);
        Element.eleReel   :=StrToFloatDef (SubItems[1],0);
        Element.eleChaine :=Caption;
      End;
      Write(Fichier,Element);
    End;
  Finally
    CloseFile(Fichier);
  End;
end;
```

Puis sous Delphi 8 en utilisant la classe TFileElement :

code Delphi 8
```
procedure TForm1.btnSauverClick(Sender: TObject);
Var Fichier:TFileElement;
    i       :Integer;
    Element:TElement;
begin
  Fichier:=TFileElement.Create;
  Try
    Fichier.AssignFile(ExtractFilePath(Application.ExeName)+'fichier.dat');
    Fichier.ReWrite;
    For i:=0 To Liste.Items.Count-1 Do
    Begin
      With Liste.Items[i] Do
      Begin
        Element.eleEntier :=StrToIntDef   (SubItems[0],0);
        Element.eleReel   :=StrToFloatDef (SubItems[1],0);
        Element.eleChaine :=Caption;
      End;
      Fichier.Write(Element);
    End;
  Finally
    Fichier.CloseFile;
    Fichier.Free;
  End;
end;
```

II-E - Acces direct à un élément

Les fichiers typés présentent l'avantage de pourvoir accéder directement à un élément particulier du fichier pour le lire ou le mettre à jour.

Cette possibilité est conservée dans la classe TFileElement. Il faut dans ce cas que la propriété Taille soit correctement renseignée.

Par exemple dans Delphi 7 pour aller lire l'élément d'index Numero dans le fichier on utilise le code suivant :

code Delphi 7
```
procedure TForm1.Button1Click(Sender: TObject);
Var Fichier:File Of TElement;
begin
  AssignFile(Fichier, ExtractFilePath(Application.ExeName)+'fichier.dat');
  Reset(Fichier);
  Try
    Seek(Fichier,Numero);
    Read(Fichier,Element);
  Finally
    CloseFile(Fichier);
  end;
end;
```

Ceci sera traduit simplement dans Delphi 8 par :

code Delphi 8

```
procedure TForm1.Button1Click(Sender: TObject);
Var Fichier:TFileElement;
begin
  Fichier:=TFileElement.Create;
  Try
    Fichier.AssignFile(ExtractFilePath(Application.ExeName)+'fichier.dat');
    Fichier.Reset;
    Fichier.Seek(Numero);
    Fichier.Read(Element);
  Finally
    CloseFile(Fichier);
    Fichier.Free;
  end;
end;
```

Conclusion

La gestion des fichiers reste donc une chose possible sous Delphi 8. Cela n'est cependant pas un moyen de stockage naturel. Il reste préférable d'utiliser des strutures de type base de données pour les volumes importants ou des fichiers de type XML voire des fichiers INI pour les petits volumes de données.

Fichiers sources de cet article :
Miroir 1 : **Source Delphi 7 de l'exemple**
Miroir 1 : **Source Delphi 8 de l'exemple**
Dans le cas où le miroir 1 ne fonctionne pas :
Miroir 2 : **Source Delphi 7 de l'exemple**
Miroir 2 : **Source Delphi 8 de l'exemple**

Merci à **Pierre Castelain** pour la correction orthographique.