

Utilisation du correcteur orthographique de Word

par [Nono40](#)

Date de publication : 14/07/2005

Dernière mise à jour : 31/07/2005

Utilisation du correcteur orthographique de Microsoft Word dans une application Delphi. Les exemples présentés ici sont testés avec Delphi 2005 et Word 2003.

- I - Introduction
- II - Avant de commencer
- III - Première méthode : WordApplication.CheckSpelling
 - III-A - Début du projet
 - III-B - Objets word utilisés
 - III-C - Principe de gestion de la correction
 - III-D - Tester et corriger chaque mot
 - III-E - Ajouter 'remplacer tout' et 'ignorer tout'
 - III-F - Changer la langue de vérification
- IV - Seconde méthode : utiliser la boîte de dialogue de Word
 - III-A - Début du projet
 - IV-B - Objets word utilisés
 - IV-C - Corriger le texte
 - IV-D - Changer la langue de vérification
- V - Gestion des dictionnaires
 - V-A - Ajouter un dictionnaire personnalisé
 - V-B - Associer un dictionnaire perso à une langue
 - V-C - Connaître la liste des dictionnaires disponibles
- VI - Téléchargement et liens

I - Introduction

Il est parfois nécessaire d'utiliser un correcteur orthographique dans une application. Les méthodes proposées ici vont utiliser le correcteur intégré à Microsoft Word.

Deux approches bien différentes seront présentées par la suite.

La première utilise directement les fonctions de test et de proposition de correction orthographique de Word. Cette méthode permet de maîtriser complètement le processus de correction, la gestion des remplacements et la gestion du dictionnaire sera manuelle. Elle doit être préférée dans le cas où le texte à analyser ne peut pas être corrigé dans sa globalité. Un exemple concret est la correction d'un fichier XML ou HTML dans lequel il ne faut pas corriger les balises et entêtes.

La deuxième méthode utilise la boîte de dialogue de correction de Word. Ici il n'est pas possible de filtrer les parties de texte à corriger. En revanche, toutes les actions de remplacement et d'ajout au dictionnaire sont gérées automatiquement. Cette méthode est préférable dans le cas de correction de textes comme des mails ou des fichiers texte.

II - Avant de commencer

Pour utiliser l'une ou l'autre des deux méthodes il faut commencer par créer et importer l'unité de bibliothèque de type.

Dans le menu composants, choisissez "*Importer un composant...*" puis choisir l'option "importer une bibliothèque de type" :

Dans la liste des bibliothèques de type disponibles, choisir "Microsoft Word 11.0 Object Library"

Dans la fenêtre suivante, choisissez le répertoire d'enregistrement de la nouvelle unité.

Puis choisissez de créer l'unité. Cette option permet de garder l'unité disponible pour tous les projets. Dans le cas contraire (ajouter au projet), l'unité ne sera créée et ajoutée que pour le projet en cours.

Vous aurez sans doute une erreur de compilation de cette unité :

Erreur : Word_TLB.pas(34713): E2015 Operator not applicable to this operand type

Cette erreur est sur la ligne :

```
IUnknown(TVarData(Params[0]).VPointer) as Range {const Range},
```

Il faut la modifier comme suit :

```
IUnknown(TVarData(Params[0]).VPointer) as Word_TLB.Range {const Range},
```

Voilà l'unité de type créée, pour l'utiliser ajoutez Word_tlb dans la clause uses.

III - Première méthode : WordApplication.CheckSpelling

Cette méthode utilise un test et une correction mot à mot. On garde un contrôle complet sur les mots testés, les corrections proposées et la gestion du dictionnaire perso.

En revanche, le traitement est plus compliqué et l'exécution plus longue.

III-A - Début du projet

Dans un nouveau projet, ajoutez une nouvelle fenêtre qui servira pour la gestion de la correction. Donnez-lui l'aspect suivant :

Dans cette fenêtre créez une nouvelle méthode publique prenant en paramètre un TRichEdit qui contiendra le texte à corriger.

```
type
TFenCorrecteur = class(TForm)
private
{ Déclarations privées }
public
{ Déclarations publiques }
Procedure Verifie(Texte:TRichEdit);
end;
```

III-B - Objets word utilisés

Pour mettre cette méthode en oeuvre il nous faut déjà créer un objet **WordApplication**. Une fois connecté à Word il faudra créer un document vide. Bien que ce document ne soit pas utilisé, il est indispensable aux fonctions de correction de Word. Voici alors le début de la méthode *Verifie* :

```
procedure TFenCorrecteur.Verifie(Texte: TRichEdit);
Var Word:TWordApplication;
    Doc :_Document;
    SaveChanges:OleVariant;
begin
Word := TWordApplication.Create(Self);
Try
// Ouverture de word et création d'un document vide
Word.ConnectKind := ckNewInstance;
Word.Connect;
Doc:=Word.Documents.Add(EmptyParam, EmptyParam, EmptyParam, EmptyParam);

Finally
SaveChanges := wdDoNotSaveChanges;
Word.Quit(SaveChanges);
Word.Free;
End;
end;
```

Notez l'utilisation de ConnectKind afin de créer une nouvelle instance de Word dans tous les cas même s'il est déjà ouvert. Ceci permet d'être totalement indépendant des instances déjà ouvertes, et

en particulier de demander la fermeture de Word sans se soucier des documents déjà ouverts.

III-C - Principe de gestion de la correction

Le texte à corriger doit être découpé en mots puis tous les mots testés un par un. De plus il faut pouvoir stopper la correction et la reprendre facilement.

L'algorithme suivant est choisi :

```

Début : position actuelle
Fin : fin du document
Si début correction en milieu de mot, aller en fin de mot
Tant que pas fin de texte
  Chercher le prochain mot
  Tester/Corriger le mot
  Tester fin de document
  Si (début<>début document)Et(fin document atteinte )
    Demander si recommencer depuis début, si oui :
    Fin : début
    Début : début document
Fin correction

```

Ceci donne le code suivant :

```

procedure TFenCorrecteur.Verifie(Texte: TRichEdit);
Var Word:TWordApplication;
    Doc :_Document;
    SaveChanges:OleVariant;
begin
Word := TWordApplication.Create(Self);
Try
  // Ouverture de word et création d'un document vide
  Word.Connect;
  Doc:=Word.Documents.Add(EmptyParam, EmptyParam, EmptyParam, EmptyParam);

  // Il faut balayer le document pour en séparer les mots.
  Chaine := Texte.Text;
  DebutCorrection := Texte.SelStart;
  i := DebutCorrection;
  FinCorrection := Length(Chaine);

  // Si on commence en milieu de mot on va en fin de mot de suite.
  While (i<FinCorrection)And(Chaine[i]In Lettres) Do Inc(i);

  // Recherche et test de tous les mots du document
  While i<FinCorrection Do
  Begin
  // Une lettre signale un début de mot
  If Chaine[i]In Lettres Then
  Begin
  Debut := i;
  Mot := Chaine[i];
  Inc(i);
  While (i<=Length(Chaine))And(Chaine[i]In LettresSigne) Do
  Begin
  Mot := Mot + Chaine[i];
  Inc(i);
  End;
  Fin := i;
  // Ici le mot est isolé, commence alors la vérification
  //.....
  End
  Else Inc(i);
  // Test de fin de document quand on est pas parti du début
  If (i>=FinCorrection)And(DebutCorrection>1) Then

```

```

Begin
  If MessageDlg('La correction à atteint la fin du document, voulez vous-corriger
le début ?'
               ,mtConfirmation,[mbYes,mbNo],0)=mrYes Then
    Begin
      FinCorrection := DebutCorrection-1;
      DebutCorrection:=1;
      i:=DebutCorrection;
    End;
  End;
  End;
  ShowMessage('Vérification terminée.');
```

```

Finally
  SaveChanges := wdDoNotSaveChanges;
  Word.Quit(SaveChanges);
  Word.Free;
End;
end;
```



Dans le cas où tous les mots de doivent pas être corrigés, il faut compléter cette procédure et ajouter des conditions sur la correction du mot.

En cas de mot mal orthographié, il faut afficher la fenêtre avec la liste des propositions puis attendre la réponse de l'utilisateur. Dans la fenêtre des propositions, placez la propriété **modalresult** de tous les boutons à **mrOk** (sauf celui d'affichage du dictionnaire perso) puis associez les évènements suivants :

```

Const
  corAbandon           = 0;
  corRemplacer         = 1;
  corRemplacerTout   = 2;
  corIgnorer           = 3;
  corIgnorerTout     = 4;
  corAjouterDico      = 5;

procedure TFenCorrecteur.btnAbandonClick(Sender: TObject);
begin
  Tag := corAbandon;
end;

procedure TFenCorrecteur.btnRemplacerClick(Sender: TObject);
begin
  Tag := corRemplacer;
end;

procedure TFenCorrecteur.btnToutRemplacerClick(Sender: TObject);
begin
  Tag := corRemplacerTout;
end;

procedure TFenCorrecteur.btnIgnorerClick(Sender: TObject);
begin
  Tag := corIgnorer;
end;

procedure TFenCorrecteur.btnToutIgnorerClick(Sender: TObject);
begin
  Tag := corIgnorerTout;
end;

procedure TFenCorrecteur.btnAjoutDicoClick(Sender: TObject);
begin
  Tag := corAjouterDico;
end;
```

III-D - Tester et corriger chaque mot

Le texte est maintenant bien découpé en mots, pour tester l'orthographe il suffit d'appeler la méthode **Word.CheckSpelling()**. Cette méthode prend un mot en paramètre et retourne True si l'orthographe est correcte. On peut difficilement faire plus simple...

Si le mot n'est pas correctement orthographié, la méthode **Word.GetSpellingSuggestions()** permet d'en connaître la liste de suggestion. Cette méthode retourne un objet **SpellingSuggestions**.

SpellingSuggestions.Count contient le nombre de suggestions possibles.

SpellingSuggestions.Item(N).Name contient la Nième suggestion.



SpellingSuggestions.Item(N) est indexé à partir de 1 et non de 0, la boucle doit donc parcourir de 1 à SpellingSuggestions.Count !

Le code permettant d'afficher la liste des suggestions est le suivant :

```
// On prépare la fenêtre de demande de correction
eInconnu.Text := Mot;
eListe.Items.Clear;
Sugg := Word.GetSpellingSuggestions(Mot);
For j:=1 To Sugg.Count Do
  eListe.Items.Add(Sugg.Item(j).Name);
If Sugg.Count>=1 Then
  Begin
    eRemplacer.Text := Sugg.Item(1).Name;
    eListe.Enabled := True;
  End Else
  Begin
    eRemplacer.Text := Mot;
    eListe.Items.Add('Pas de suggestion');
    eListe.Enabled := True;
  End;
```

Le remplacement le plus probable est toujours donné en premier de la liste, par défaut le champ "remplacé par" sera donc rempli par la première possibilité.

Il suffit alors de sélectionner le mot en question dans le richedit puis d'appeler la fenêtre de correction. Le code minimal de correction devient :

```
// On se place dans RichEdit au bon endroit
Texte.SelStart := Debut-1;
Texte.SelLength := Fin-Debut;

// affichage de la fenêtre et traitement
Tag := corAbandon;
ShowModal;
Case Tag Of
  corAbandon : Exit; // Annuler/Abandon sort de la procédure
  corIgnorer :; // ignore ne fait rien et passe au mot suivant
  corRemplacer:Begin // On change il faut alors modifier le texte
    // Remplacement dans le R
    Texte.SelText := eRemplacer.Text;
    // Remplacement dans la chaîne (pour que les positions correspondent)
    Delete(Chaine,Debut,Fin-Debut);
    Insert(eRemplacer.Text,Chaine,Debut);
    i:=Debut+Length(eRemplacer.Text);
  End;
End;
```

III-E - Ajouter 'remplacer tout' et 'ignorer tout'

Nous allons garder les mots à ignorer et ceux à remplacer dans deux listes distinctes.

Quand un nouveau mot à corriger est rencontré, on teste s'il fait partie de la liste 'ignorer tout'. Si oui, on passe au mot suivant.

Ensuite on teste s'il fait partie de la liste 'remplacer tout'. Si oui on effectue le remplacement automatiquement.

```
// Ici le mot est isolé, commence alors la vérification
If Not Word.CheckSpelling(Mot) Then
Begin
// On teste déjà s'il ne fait pas partie des mots à ignorer cette fois-ci
If ListeIgnore.IndexOf(Mot)<0 Then
Begin
// On teste ensuite si le mot n'est pas dans la liste des "tout remplacer"
If ListeRemplace.IndexOfName(Mot)<0 Then
Begin
// ...
// Traitement normal
// ...
End Else
Begin // Le mot fait partie de "tout remplacer", on le fait donc directement
// On se place dans le RichEdit au bon endroit
Texte.SelStart := Debut-1;
Texte.SelLength := Fin-Debut;
// Remplacement dans le RichEdit
Texte.SelText := ListeRemplace.Values[Mot];
// Remplacement dans la chaîne (pour que les positions correspondent)
Delete(Chaîne,Debut,Fin-Debut);
Insert(ListeRemplace.Values[Mot],Chaîne,Debut);
i:=Debut+Length(ListeRemplace.Values[Mot]);
End;
End;
End;
```

Bien sûr il faut remplir les listes au fur et à mesure des demandes :

```
Case Tag Of
corAbandon : Exit; // Annuler/Abandon sort de la procédure
corIgnorer :; // ignorer ne fait rien et passe au mot suivant
corIgnorerTout :Begin // ignorer tout ajoute le mot dans la liste des mots ignorés
ListeIgnore.Add(eInconnu.Text);
End;
corRemplacerTout,
corRemplacer:Begin // On change il faut alors modifier le texte
// Remplacement dans le RichEdit
Texte.SelText := eRemplacer.Text;
// Remplacement dans la chaîne (pour que les positions correspondent)
Delete(Chaîne,Debut,Fin-Debut);
Insert(eRemplacer.Text,Chaîne,Debut);
i:=Debut+Length(eRemplacer.Text);
// Dans le cas de tout remplacer, on garde la trace du changement sous la main
If Tag=corRemplacerTout Then
ListeRemplace.Add(Mot+'='+eRemplacer.Text);
End;
End;
```

III-F - Changer la langue de vérification

La méthode **WordApplication.CheckSpelling()** utilise le dictionnaire de la langue par défaut de Microsoft Office. Le paramètre **MainDictionary** est ignoré dans tous les cas (on peut même mettre une valeur bidon sans problème...).

Le seul moyen de changer la langue est le suivant :

- Aller dans Menu Démarrer->Programmes->Microsoft Office->Outils Microsoft Office->Paramètres linguistiques
- Dans le bas choisir la langue par défaut des applications Office
- Valider
- Il faut ensuite choisir "Confirmer et perdre les personnalisations"



Cette méthode va changer la langue par défaut de toutes les applications, de plus si Word est déjà lancé elle ne sera prise qu'au prochain démarrage.

Nous allons donc utiliser une autre méthode.

Comme la limitation de **WordApplication.CheckSpelling()** est trop pénalisante, nous allons utiliser l'objet **SpellingErrors** de l'objet **Range**. **SpellingErrors** contient la liste des erreurs du texte associé à l'objet **Range**. Si **Range** ne contient qu'un seul mot, alors **SpellingErrors** indiquera si le mot est correct ou non. De même les propositions de remplacement seront demandées sur l'objet **Range** plutôt que sur l'objet **WordApplication**.

Dans le code il faut remplacer :

```
If Not Word.CheckSpelling(Mot) Then
//...
Sugg := Word.GetSpellingSuggestions(Mot);
```

Par :

```
Word.Selection.Text := Mot;
If Word.Selection.Range.SpellingErrors.Count>0 Then
//...
Sugg := Word.Selection.Range.GetSpellingSuggestions( EmptyParam, EmptyParam, EmptyParam
, EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam
, EmptyParam, EmptyParam, EmptyParam, EmptyParam, EmptyParam );
```

Maintenant pour choisir la langue, il faut fixer en début de vérification la langue par défaut de l'objet **WordApplication.Selection**. Ceci n'a besoin d'être fait qu'une seule fois pour toute la vérification, mais peut être modifié en cours en cas de besoin.

```
Word.Selection.LanguageID := wdEnglishUS;
// ou
Word.Selection.LanguageID := wdFrench;
```



Il faut bien sûr que le dictionnaire correspondant soit installé dans Word pour que la vérification fonctionne.

IV - Seconde méthode : utiliser la boîte de dialogue de Word

Cette méthode va traiter le texte dans sa globalité. Il y a beaucoup moins de contrôle possible sur la gestion de la correction. En revanche le code est nettement plus simple et la correction grammaticale est aussi effectuée.

III-A - Début du projet

Dans un nouveau projet, ajoutez une nouvelle unité simple qui servira pour la gestion de la correction.

Dans cette unité créez une nouvelle procédure publique prenant un paramètre un TRichEdit qui contiendra le texte à corriger.

```
unit UCorrecteur;

interface

Uses ComCtrls;

Procedure Verifie(Texte: TRichEdit);

implementation

Uses Classes, Forms, Variants, SysUtils, Dialogs, Word_TLB, OleServer;

procedure Verifie(Texte: TRichEdit);
begin
  //...
end;
```

IV-B - Objets word utilisés

Pour mettre cette méthode en oeuvre il nous faut déjà créer un objet **WordApplication**. Une fois connecté à Word il faudra créer un document vide. Bien que ce document ne soit pas utilisé, il est indispensable aux fonctions de correction de Word. Voici alors le début de la méthode *Verifie* :

```
procedure Verifie(Texte: TRichEdit);
Var Word: TWordApplication;
    Doc :_Document;
    SaveChanges: OleVariant;
begin
  Word := TWordApplication.Create(Application);
  Try
    // Ouverture de word et création d'un document vide
    Word.ConnectKind := ckNewInstance;
    Word.Connect;
    Doc:=Word.Documents.Add(EmptyParam, EmptyParam, EmptyParam, EmptyParam);

  Finally
    SaveChanges := wdDoNotSaveChanges;
    Word.Quit(SaveChanges);
    Word.Free;
  End;
end;
```

Notez l'utilisation de `ConnectKind` afin de créer une nouvelle instance de Word dans tous les cas même s'il est déjà ouvert. Ceci permet d'être totalement indépendant des instances déjà ouvertes, et en particulier de demander la fermeture de Word sans se soucier des documents déjà ouverts.

IV-C - Corriger le texte

La méthode choisie ici est simple, le texte à corriger va être placé dans le document ouvert. Puis nous allons appeler la fenêtre de gestion de la correction de Word. Il se chargera lui-même de toutes les opérations liées à la correction.

En fin de traitement il n'y a plus qu'à récupérer le texte corrigé et à le placer de nouveau dans le RichEdit.

Pour appeler une des boîtes de dialogue standards de Word il faut utiliser la collection **WordApplication.Dialogs**. Cette collection est indexée par une liste d'identificateurs, celui qui nous intéresse est **wdDialogToolsSpellingAndGrammar**.

Il suffit ensuite d'appeler la méthode **Show** de l'objet boîte de dialogue. Tout la correction sera alors gérée par Word.

La méthode **Show** retourne une valeur de type **Integer** permettant de connaître la cause de fermeture de la boîte de dialogue.

Valeur	cause de fermeture
-2	Sortie par le bouton "fermer"
-1	Sortie par le bouton "OK"
0	Sortie par le bouton annuler
>0	Sortie par l'un des autres boutons

La valeur qui nous intéresse ici est "0", sortie par annulation.

Le code complet de la correction devient :

```

procedure Verifie(Texte: TRichEdit);
Var Word:TWordApplication;
    Doc :_Document;
    SaveChanges:OleVariant;
begin
  Word := TWordApplication.Create(Application);
  Try
    // Ouverture de word et création d'un document vide
    Word.ConnectKind := ckNewInstance;
    Word.Connect;
    Doc:=Word.Documents.Add(EmptyParam, EmptyParam, EmptyParam, EmptyParam);

    // copie du texte dans Word
    Doc.Content.Text := Texte.Text;

    // Appel de la fenêtre de correction de Word, le texte modifié est contenu dans
    // l'objet _Document, on n'a pas de contrôle sur ce qui se passe.
    // On récupère le texte modifié par Word sauf si la correction est annulée
    If Word.Dialogs.Item(wdDialogToolsSpellingAndGrammar).Show(EmptyParam)<>0
    Then Texte.Text := Doc.Content.Text;
  
```

```
Finally
  SaveChanges := wdDoNotSaveChanges;
  Word.Quit(SaveChanges);
  Word.Free;
End;
ShowMessage('Vérification terminée.');
```

IV-D - Changer la langue de vérification

C'est ici très simple de la changer, il faut fixer la langue par défaut de l'objet **Document.Content**.

```
// Sélection de la langue de correction
Doc.Content.LanguageID := wdEnglishUS;
// Ou
Doc.Content.LanguageID := wdFrench;
```

Ce code doit être placé juste avant l'affectation du texte à **Document.Content.Text**.

V - Gestion des dictionnaires

V-A - Ajouter un dictionnaire personnalisé

Un dictionnaire est un simple fichier texte contenant un mot par ligne. Pour ajouter un mot il suffit de mettre à jour le fichier.

Dans l'exemple nous allons ajouter un dictionnaire nommé *MonDicoPerso.dic* et le placer là où se trouve l'exécutable.

Il est souvent très utile d'avoir un dictionnaire pour y ajouter les mots courants. L'objet **WordApplication** dispose d'une propriété **CustomDictionaries**. Cette propriété est une collection d'objets **Dictionary**.

La propriété **CustomDictionaries.Item()** permet d'atteindre les dictionnaires un par un. L'ajout d'un dictionnaire à cette liste est permanente tant qu'il n'est pas explicitement supprimé. Au début de la correction il faut tester la présence du dictionnaire personnalisé et ne l'ajouter qu'en cas de besoin.

En appliquant les points ci-dessus voici le code pour créer, lire et lier le dictionnaire :

```
// Test d'existence du dico personnalisé
// Chargement/création suivant le cas
NomDico := ExtractFilePath(ParamStr(0))+MonDico;
If FileExists(NomDico)
  Then ListeDico.LoadFromFile(NomDico)
  Else ListeDico.SaveToFile(NomDico);

// Ajout du dico personnalisé
Dicos := Word.CustomDictionaries;
Trouve:=False;
For i:= 1 To Dicous.Count Do
Begin
  Index:= i;
  If UpperCase(Dicos.Item(Index).Name)=UpperCase(MonDico) Then Trouve:=True;
End;
If Not Trouve Then
  Dicous.Add(NomDico);
```

Dans le code de gestion de la première méthode il faut prendre en compte le choix 'ajouter au dictionnaire' :

```
// affichage de la fenêtre et traitement
Tag := corAbandon;
ShowModal;
Case Tag Of
  //...
  //...
  corAjouterDico:Begin
    // On met à jour le fichier .dic
    ListeDico.Add(eInconnu.Text);
    ListeDico.SaveToFile(NomDico);
  End;
End;
```

Il n'est pas nécessaire de supprimer puis de lier à nouveau le dictionnaire dans Word. Il sera automatiquement pris en compte à la prochaine demande de correction.

Le code du bouton pour afficher le dictionnaire est simplement un **ShellExecute** appelant NotePad :

```
procedure TFenCorrecteur.btnVoirDicoClick(Sender: TObject);
begin
  ShellExecute(0, 'OPEN', PChar(ExtractFilePath(ParamStr(0))+MonDico)
    , Nil, Nil, SW_SHOWNORMAL);
end;
```

V-B - Associer un dictionnaire perso à une langue

Dans la section précédente le dictionnaire personnalisé ne dépend pas de la langue. Si vous voulez le rendre dépendant d'une langue il faut ajouter le code suivant une fois le dictionnaire ajouté à la liste :

```
// On utilise le nom de fichier comme index de recherche
Index := ExtractFilePath(ParamStr(0))+MonDico;
Dico := Word.CustomDictionaries.Item(Index);
// Le dictionnaire ne sera utilisé que pour les corrections FR
Dico.LanguageSpecific := True;
Dico.LanguageID := wdFrench;
```

V-C - Connaître la liste des dictionnaires disponibles

Avant de demander une correction dans une langue particulière, il faut s'assurer qu'il existe un dictionnaire principal associé.

La collection **WordApplication.Languages** donne la liste des langues utilisables dans Word. Pour chacune des langues il est possible de connaître le dictionnaire associé s'il existe.

Cette collection est indexée sur les identificateurs des langues (wdxxxx). Elle ne contient pas d'index ordinal de 1 à **Languages.Count**. C'est un des rares cas où VB à un avantage sur Delphi, en effet en VB l'instruction **For Each** permet de balayer une collection sans en connaître d'index. En delphi il faut utiliser l'énumérateur de collection **_NewEnum** de l'objet collection afin de connaître tous les éléments sans en connaître d'index. La méthode **_NewEnum.Next** permet d'extraire un certain nombre d'éléments de la collection, ici ils seront lus un par un.

Le code suivant remplit un TLisView avec la liste des langues utilisables :

```
procedure TFenLangues.FormShow(Sender: TObject);
Var Word:TWordApplication;
    Langs:Languages;
    SaveChanges:OleVariant;
    IEnum :IEnumVariant;
    Nombre : Cardinal;
    Element : OleVariant;
    langue : language;
    IDisp : IDispatch;
begin
  Word := TWordApplication.Create(Self);
  Try
    Langs:=Word.Languages;
    IEnum:=Langs._NewEnum as IEnumVariant;
    While IEnum.Next(1,Element,Nombre)=S_OK Do
    Begin
      // Indispensable ici de passer par un IDisp intermédiaire...
      IDisp := Element;
      Langue := IDisp As Language;
```

```
// MAJ de la liste
With Liste.Items.Add Do
Begin
Caption := IntToStr(Langue.ID);
SubItems.Add(Langue.Name);
SubItems.Add(Langue.NameLocal);
If Langue.ActiveSpellingDictionary=Nil
Then SubItems.Add('Non')
Else SubItems.Add('Oui');
If Langue.ActiveGrammarDictionary=Nil
Then SubItems.Add('Non')
Else SubItems.Add('Oui');
End;
End;
Finally
SaveChanges := wdDoNotSaveChanges;
Word.Quit(SaveChanges);
Word.Free;
End;
end;
```

VI - Téléchargement et liens

Fichiers sources de cet article :

Miroir 1 : [Sources des exemples \[11Ko\]](#)

Dans le cas où le miroir 1 ne fonctionne pas :

Miroir 2 : [Sources des exemples \[11Ko\]](#)

Version PDF de cet article :

Miroir 1 : [Version PDF](#)

Dans le cas où le miroir 1 ne fonctionne pas :

Miroir 2 : [Version PDF](#)

[Utiliser Word dans Delphi](#)

MSDN : [CheckSpelling Method](#)

MSDN : [Displaying Built-in Word Dialog Boxes](#)

[More Automation In Delphi](#)

Merci à [Olivier Lance](#) et [Laurent Dardenne](#) pour leurs remarques et la correction orthographique.