

# Comment passer un nombre variable de paramètres à une procédure ?

par [Laurent Dardenne \(Contributions\)](#)

Date de publication : 03/09/2007

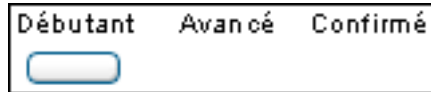
Dernière mise à jour : 03/09/2007

Il est parfois utile de pouvoir passer un nombre inconnu de paramètres à une procédure. A l'origine cette astuce devait rejoindre la FAQ Delphi mais les explications étant suffisamment nombreuses il a été décidé de la présenter sous forme de tutoriel.

Je tiens à remercier **Waskol** pour son rappel sur l'usage des tableaux de variant.

- 1 - Public concerné
  - 1-1 - Les sources
- 2 - L'énoncé du problème
- 3 - Solution 1 : Le type Variant
  - 3-1 - Tableau de variant
- 4 - Solution 2 : Le type TVarRec
  - 4-1 - Problème de reconnaissance de type
- 5 - Comment créer une fonction pouvant renvoyer un résultat de type quelconque ?
- 6 - Delphi .NET
- 7 - Liste des types d'un Variant et d'un TVarRec
- 8 - Conclusions

## 1 - Public concerné



Testé sous XP sp2 et Delphi 2006 sp10.

### 1-1 - Les sources

Les fichiers sources des différents exemples :

**FTP.**

**HTTP.**

## 2 - L'énoncé du problème

Quelquefois on peut être amené à manipuler des DLL écrites en C ou C++ utilisant un nombre variable de paramètres, dans ce cas la solution se trouve [FAQ ici](#).

En revanche de réaliser en natif une méthode utilisant un nombre variable de paramètres nécessite une autre approche. Dans un premier temps on pourrait penser qu'un tableau de pointeur suffirait mais au moins 2 questions se posent, à savoir comment connaître :

- le nombre de paramètres dans un tableau de pointeurs ?
- retrouver le type de chaque paramètre ?

Les traitements à mettre en place seraient assez délicats. Le langage Delphi nous proposant le type **Variant**, pouvant encapsuler n'importe quel type Delphi, voyons comment procéder sans trop d'effort.

### 3 - Solution 1 : Le type Variant

Projet : ..\TypeTvarRec\ParametresVariables1

Il est donc possible, en utilisant **le type Variant**, de passer un paramètre pouvant être de différent type. Attention ils ne sont pas tous supportés.

```
procedure ParametreVariant(AValeur:Variant);
begin
  WriteLn('Un seul element du type ',VarTypeAsText(VarType(Variant(AValeur))),
    ' sa valeur est ',Variant(AValeur));
end;
```

Les appels suivant sont valides :

```
ParametreVariant('Une chaine');
ParametreVariant(3.14159);
ParametreVariant(True);
```

Et renvoient :

```
Un seul element du type String sa valeur est Une chaine
Un seul element du type Double sa valeur est 3,14159
Un seul element du type Boolean sa valeur est True
```

Un **Variant** pouvant être un tableau, améliorons notre procédure pour gérer ce cas :

```
procedure ParametreVariant(AValeur:Variant);
var I      : Integer;
    LimiteHaute,
    LimiteBasse : Integer;
begin
  if VarIsArray(AValeur) then
  begin
    LimiteHaute := VarArrayHighBound(AValeur, 1);
    LimiteBasse := VarArrayLowBound(AValeur, 1);
    for I := LimiteBasse to LimiteHaute do
      WriteLn('L'element ', I,
        ' est du type ',VarTypeAsText(VarType(Variant(AValeur[i]))),
        ' sa valeur est ',Variant(AValeur[i]));
    ReadLn;
  end
  else
  begin
    WriteLn('Un seul element du type ',VarTypeAsText(VarType(Variant(AValeur))),
      ' sa valeur est ',Variant(AValeur));
  end;
end;
```

#### 3-1 - Tableau de variant

La lecture du code précédent nous indique qu'un **Variant** peut contenir un type tableau, l'appel, en construisant statiquement le tableau à l'aide de la procédure **VarArrayOf**, devient dans ce cas :

```
// Déclare et initialise qq variables
Var Chaine1 : String='Une chaine';
Integer1 : Integer=90;
Double1 : Double=5.6;
Double2 : Double=3.14159;
Boolean1 : Boolean=True;
Chaine2 : String='s';

begin
// Passage d'un nombre de paramètre variable, uniquement des constantes
ParametreVariant(VarArrayOf(['Une chaine', 90, 5.6, 3.14159, True, 's']));
```

Le paramètre tableau ouvert de **Variant**, **array of Variant**, permet de transmettre un tableau d'expressions de différents types :

```
procedure TableauVariant(AValeur:array of Variant);
var I : Integer;
begin
for i := Low(AValeur) to High(AValeur) do
  WriteLn('L'element ', I,
    ' est du type ',VarTypeAsText(VarType(Variant(AValeur[i]))),
    ' sa valeur est ',Variant(AValeur[i]));
ReadLn;
end;
```

Comme nous l'avons précédemment écrit tous les types ne sont pas supportés. La gestion d'une instance d'objet est un peu particulière car on doit le transtyper, plus précisément son adresse, en **LongWord** :

```
//Utilisation d'une variable de type tableau de variant
TabVariant:=VarArrayOf(['Une chaine', 90, 5.6, 3.14159, True, 's',LongWord(UnObjet)]);
TableauVariantV2(TabVariant);
```

On manipule le **LongWord** reçu, dont le code Variant est **varLongWord**, de la manière suivante :

```
procedure TableauVariantV2(A: array of Variant);
//Affiche le type et la valeur des occurrences du tableau de variants.
var I : Integer;
Objet : TObject;

begin
for I := Low(A) to High(A) do
  begin
    WriteLn('L'élément ', I, ' est du type ',VarTypeAsText(VarType(Variant(A[i]))),' sa valeur est '
    ,Variant(A[i]));
    //Une référence d'objet est passé en tant que LongWord
    if VarIsType(A[i],varLongWord) then
      begin
        Writeln(Format('Adresse %x',[LongWord(A[i])]));
        try //Est-ce bien une référence d'objet valide ?
          Objet:=Pointer(LongWord(A[i])); //Récupère une référence à partir du variant
          Writeln('Objet '+Objet As TObject).ClassName); //Tentative d'accès à l'objet
        Except
          on EAccessViolation do //Le variant n'héberge pas une référence d'objet valide.
            Writeln('Erreur de transtypage');
          end;
      end;
  end;
```

```
end;  
end;  
ReadLn;  
end;
```

La procédure **VarType** renvoie le code du type contenu dans un variant, on peut tester directement un type donné en utilisant la procédure **VarIsType**.

La documentation de Delphi, à propos du tableau de variant, nous indique :

*"Les paramètres tableau ouvert variant permettent de transmettre un tableau d'expressions de types différents à une seule routine. Pour définir une routine utilisant un paramètre tableau ouvert variant, spécifiez `array of const` comme type du paramètre. Donc"*

```
procedure DoSomething(A: array of const);
```

déclare une procédure appelée **DoSomething** qui peut agir sur des tableaux de données hétérogènes.

La construction **array of const** est équivalente à **array of TVarRec**. **TVarRec**, déclaré dans l'unité `System`, représente un enregistrement avec une partie variable qui peut contenir des valeurs de type entier, booléen, caractère, réel, chaîne, pointeur, classe, référence de classe, interface et variant. Le champ `VType` de **TVarRec** indique le type de chaque élément du tableau. Certains types sont transmis comme pointeur et non comme valeur ; en particulier les chaînes longues sont transmises comme **Pointer** et doivent être transtypées en **String**.

L'exemple suivant utilise un paramètre tableau ouvert **Variant** dans une fonction qui crée une représentation sous forme de chaîne de chaque élément transmis et concatène le résultat dans une seule chaîne. Les routines de manipulation de chaînes utilisées dans cette fonction sont définies dans l'unité `SysUtils`.

```
function MakeStr(const Args: array of const): string  
var  
  I: Integer;  
begin  
  Result := '';  
  for I := 0 to High(Args) do  
    with Args[I] do  
      case VType of  
        vtInteger: Result := Result + IntToStr(VInteger);  
        vtBoolean: Result := Result + BoolToStr(VBoolean);  
        vtChar: Result := Result + VChar;  
        vtExtended: Result := Result + FloatToStr(VExtended^);  
        vtString: Result := Result + VString^;  
        vtPChar: Result := Result + VPChar;  
        vtObject: Result := Result + VObject.ClassName;  
        vtClass: Result := Result + VClass.ClassName;  
        vtAnsiString: Result := Result + string(VAnsiString);  
        vtCurrency: Result := Result + CurrToStr(VCurrency^);  
        vtVariant: Result := Result + string(VVariant^);  
        vtInt64: Result := Result + IntToStr(VInt64^);  
      end;  
    end;  
  end;  
end;
```

Il est possible d'appeler cette fonction en utilisant un constructeur de tableau ouvert. Par exemple,

```
MakeStr(['test', 100, ' ', True, 3.14159, TForm]);
```

*renvoie la chaîne 'test100 -13.14159TForm'.*

Vous remarquerez que les codes des types manipulés diffèrent entre un **Variant** et un **TVarRec**, et il n'est plus nécessaire lors de l'appel de transtyper une instance d'objet.


Les variants étant coûteux en temps de traitement et dédiés aux applications COM, dans les autres cas il est préférable d'utiliser le type **TVarRec**.

## 4 - Solution 2 : Le type TVarRec

Projet : ..\TypeTvarRec\ParametresVariables3

Examinons notre question initiale au travers de l'exercice suivant, créer un tableau dynamique et le parcourir au travers des itérateurs **For..in..Do**. La construction suivante n'étant pas autorisée :

```
For MaVariable in [Objet1,Objet2,Objetn] do
```

 Les utilisateurs de Delphi 7 et inférieure devront utiliser une boucle classique pour itérer le tableau construit.

Déclarons une classe basique et un type de tableau dynamique :

```
{$ASSERTIONS ON}
type
TMaClasse=class
end;

TMaClasseDynArray = array of TMaClasse;
```

Notre procédure spécifie **array of const** comme type du paramètre :

```
Function MakeArray(Args: Array of const) :TMaClasseDynArray;
//Construit un tableau d'un type spécifique
Var I: Integer;
    Count : Integer;
begin
    Count:=High(Args);
    Assert(Count>0,'Tableau vide non supporté.');
```

SetLength(Result,Count+1);

```
for I := 0 to High(Args) do
with Args[i] do
    case VType of
        vtObject: Result[i]:= TMaClasse(VObject); //Implique un tanstypage pour chaque type de
tableau manipulé
        else Assert(1<>1,'Type non supporté.');
```

end;//Case

```
end;
```

Nous limitant à la manipulation d'un tableau contenant un seul un type de donnée, l'usage d'assertions nous avertira dans le cas contraire. Notez la simplicité du code comparé à l'utilisation de **Variants**.

```
begin
Obj1:=TMaClasse.Create;
Obj2:=TMaClasse.Create;
Obj3:=TMaClasse.Create;
try
    TabDynamique:=MakeArray([Obj1,Obj2,Obj3]);


    for Current in TabDynamique do
        Writeln(Current.ClassName);
        // Delphi 7 et <
//Current : TObject
```

```

//Writeln(TMaClasse(Current).ClassName);
Finally
  Obj1.Free;
  Obj2.Free;
  Obj3.Free;

  Finalize(TabDynamique);
end;
end.

```

 *Attention, dans notre exemple c'est à l'appelant de libérer le tableau construit.*

Il reste un problème car sous cette forme notre procédure nécessite de déclarer autant de type tableau dynamique que de type manipulé.

#### Projet : ..\TypeTvarRec\ParametresVariables4

Modifions le type de retour en utilisant le type **TVarRec**:

```

TVarRecDynArray= Array of TVarRec;
...
Function MakeArray(Args: Array of const) : TVarRecDynArray;

```

On déplace ainsi le transtypage en dehors de la méthode construisant le tableau tout en réduisant la déclaration de type de tableaux dynamique. C'est désormais à l'appelant d'indiquer le type manipulé, connu via le champ **VType** :

```

var ArrayVarRec : TVarRecDynArray;
    Current : TVarRec;
...
Try
  Try
    ArrayVarRec:=MakeArray([TObject.Create, TForm.Create(Nil), TLabel.Create(Nil) ]);
    for Current in ArrayVarRec do
      Writeln(TObject(Current.VObject).ClassName);
    except
      On EAssertionFailed do Writeln('Erreur d'assertion.');
```


- 10 -

```

    end;
  Finally
    FreeAndFinalize(ArrayVarRec);
  end;

ArrayVarRec:=MakeArray([TObject.Create, TForm.Create(Nil), TLabel.Create(Nil) ]);
Writeln(TObject(Current.VObject).ClassName);

```

 *A noter que le code précédent ne manipule que des propriétés communes, dans le cas contraire l'usage des **RTTI** peut vous aider.*

Quant à la procédure **FreeAndFinalize** elle se charge de libérer les objets créés :

```

procedure FreeAndFinalize(AArray : TVarRecDynArray);
Var Current : TVarRec;
    Objet: TObject;
begin

```

```
For Current In AArray Do
begin
  Objet:=TObject(Current.VObject);
  FreeAndNil(Objet);
end;
Finalize(AArray);
end;
```

Notez que l'usage d'une TList peut également être une solution, de plus sous BDS cette classe implémente un itérateur...

## 4-1 - Problème de reconnaissance de type

Cette solution autour du TVarRec contient un autre souci concernant les types gérés.

N'ayant pas testé tous les cas de figure autour de cette solution, **Thierry Laborde** et **Bestiol** ont rencontrés le problème suivant :

Si on utilise un *Array of variant* nous sommes capable de détecter, par le biais de la procédure **VarIsType**, un type Date.

En revanche ce n'est pas faisable avec le *Array of Const*, car cette construction nous renvoi **vtExtended** pour le type **TDateTime**. Ce qui est logique au vue de la déclaration de ce dernier type :

```
TDateTime = type Double;
```

Dans ce cas on ne peut donc différencier un **Float** d'une date alors que la solution 1, autour du **Array of Variant**, permet bien de gérer ces 2 types différents.

On peut utiliser la solution 1 pour résoudre ce problème, sauf qu'avec un *Array of variant* on ne peut pas passer un objet aisément.

Après réflexions pour résoudre ce cas on combine les 2 solutions, l'usage d'*Array of Const* dans la procédure et le passage d'un type variant dans la liste des paramètres d'appel.

Comme ceci :

```
procedure ParamVariable(Args: Array of const);
Var Current : TVarRec;
    UneDate: TDateTime;
    Objet: TObject;
    I :Integer;
begin
  //For Current In Args Do
  for I := 0 to High(Args) do
  begin
    Current:=Args[I];
    with Current do
      case VType of
        vtVariant: if not VarIsClear(VVariant^) then
```

```
        if VarIsType(VVariant^, varDate) then
        begin
            UneDate:= VarToDateTime(VVariant^);
            Writeln('Date : ',DateToStr(UneDate));
        end;
    vtObject: begin
        Objet:=TObject(Current.VObject);
        Writeln('objet : ',Objet.ClassName);
    end;

end; //case
end;
end;
```

Attention à transtyper en **Variant** la variable de type **TDateTime** lors de l'appel :

```
ParamVariable([variant(Date),Objet1 ]);
```



*L'usage d'une variable de type **variant** contenant une date n'est pas possible car le compilateur extrait la variable encapsulée avant l'appel de notre méthode.*

## 5 - Comment créer une fonction pouvant renvoyer un résultat de type quelconque ?


A la lecture de ce qui précède le simple énoncé de cette question devrait vous donner la solution.

On peut utiliser le type **variant** pour la valeur de retour d'une fonction :

```
function FonctionRetournantPlusieursTypes(const TypeParm: integer): Variant;
begin
  case TypeParm of
    1: Result := -1;
    2: Result := 3.5;
    3: Result := 'UneChaine';
    4: Result := StrToDate('04/08/2007');
  end;
end;
```

Ce qui peut parfois aider à contourner la limitation suivante de la directive **overload** :

Les routines surchargées doivent pouvoir se distinguer par le nombre ou le type de leurs paramètres.

 *Si vous combinez l'usage de cette directive et celui de **variant** dans la liste de paramètre vous risquez quelques effets de bord. Consultez l'aide en ligne pour le détail.*

On peut aussi utiliser le type **TVarRec**

```
program FonctionVariable1;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  Classes;

function Appel(Cas: Integer) : TVarRec;
var
  Termine : Boolean;
  Longueur : Integer;
  Liste : TStringList;
begin
  Termine:=True;
  Longueur:=105;
  with Result do
  case Cas of
    0 : begin
      VType:=vtBoolean;
      VBoolean:=Termine;
    end;
    1 : begin
      VType:=vtInteger;
      VInteger:=Longueur;
    end;
    2 : begin
      VType:=vtObject;
      VObject:=TStringList.Create;
      TStringList(VObject).Add('Je suis une instance de TStringList');
    end;
    3 : begin
```

```

        VType:=vtClass;
        VClass:=TObject;
    end;
end;

procedure Traite(ValeurRetour :TVarRec);
Const S='Retour de fonction avec le type : %S ';
begin
    With ValeurRetour do
    Case VType of
        vtBoolean: Writeln(Format(S,['Boolean']),VBoolean);
        vtInteger:Writeln(Format(S,['Integer']),VInteger);
        vtObject:begin
            With TStrings(VObject) do
            begin
                Writeln(Format(S,['Objet']),TStringList(VObject)[0]);
                Free;
            end;
        end;
        vtClass:Writeln(Format(S,['Classe']),VClass.ClassName);
    end;
end;

var I:Integer;
begin
    For I:=0 to 3 do
    begin
        Traite(Appel(I));
    end;
    readln;
end.

```



⚠ Notez, dans la fonction Appel, l'attribution des valeurs à l'enregistrement de retour (**Result**). On renseigne le type et la valeur.

Au lieu de passer en paramètre un numéro de type à retourner, l'usage d'un paramètre de type **TVarRec** permettrait de récupérer une valeur de type quelconque :

```

function Appel(Args: Array of const) : TVarRec;
//Cette fonction n'effectue aucun traitement si ce n'est de renvoyer le paramètre reçu
Const S='Appel de fonction avec le type : %S ';
begin
    Assert(High(Args)<>1,'Une et une seule valeur attendue. ');
    Writeln(Format(S,[cstNameOfVType[Args[0].VType]]));
    Result:=Args[0];
    {ou peut aussi construire le TVarrec à retourner
    with Result do
    case Data.VType of
        vtBoolean : begin
            VType:=vtBoolean;
            VBoolean:=Data.VBoolean;
        end;
        vtObject : begin
            VType:=vtObject;
            VObject:=TStringList.Create;
            TStrings(Result.VObject).Add('Je suis une instance de TStrings')
        end;
        vtInteger : begin
            ...
        end;
    }
end;

```

Eviter tout de même de généraliser ce type de code :-).

## 6 - Delphi .NET

Sous Delphi .NET les variants sont présent à des fins de compatibilité:


```
Variant = type TObject;  
OleVariant = type TObject;
```

Rappel de la documentation de Delphi:

*Sur la plate-forme **.NET**, un paramètre tableau ouvert variant est équivalent à **array of TObject**. Pour déterminer le type d'un élément du tableau, vous pouvez utiliser les méthodes **TObject.ClassName** ou **Object.GetType**.*

Consultez aussi la faq [FAQ Delphi .NET](#).

## 7 - Liste des types d'un Variant et d'un TVarRec


 Les correspondances de ce tableau sont à vérifier.

VarData	TVarRec
varSmallInt: (VSmallInt: SmallInt)	?
varInteger: (VInteger: Integer)	vtInteger:(VInteger: Integer; VType: Byte)
varSingle: (VSingle: Single)	?
varDouble:(VDouble: Double)	vtExtended:(VExtended: PExtended)
varCurrency: (VCurrency: Currency)	vtCurrency: (VCurrency: PCurrency)
varDate: (VDate: TDateTime)	?
?	vtPChar:(VPChar: PChar)
?	vtObject:(VObject: TObject)
?	vtClass:(VClass: TClass)
varError: (VError: HRESULT)	?
varBoolean: (VBoolean: WordBool)	vtBoolean: (VBoolean: Boolean)
	* taille différente
varUnknown: (VUnknown: Pointer)	vtInterface: (VInterface: Pointer)
varDispatch: (VDispatch: Pointer)	Idem
varShortInt: (VShortInt: ShortInt)	?
varByte: (VByte: Byte)	vtChar: (VChar: Char)
varWord: (VWord: Word)	?
varLongWord: (VLongWord: LongWord)	?
varInt64: (VInt64: Int64)	vtInt64: (VInt64: PInt64)
varString: (VString: Pointer)	vtString: (VString: PShortString)
?	vtAnsiString: (VAnsiString: Pointer)
?	vtWideString: (VWideString: Pointer)
varOleStr: (VOleStr: PWideChar)	vtWideChar: (VWideChar: WideChar)
?	vtPWideChar: (VPWideChar: PWideChar)
varAny: (VAny: Pointer)	?
varArray: (VArray: PVarArray)	vtVariant: (VVariant: PVariant)
varByRef: (VPointer: Pointer)	vtPointer: (VPointer: Pointer)

## 8 - Conclusions

Ces quelques astuces vous permettront de répondre à certains de vos besoins en attendant les génériques qui devraient être implémentés en Win32 dans Delphi "**Tiburón**"(2008) et offriront peut être des solutions plus élégantes. Faute de temps et comme je l'ai dit, tous les cas de figure n'ont pas été testés ni le sujet de **TVarRec** abordé en profondeur. Si toutefois vous avez des remarques ou des ajouts à formuler n'hésitez pas à nous contacter.

Lien :

 [Open array parameters and array of const](#) Tutoriel traitant de nombreux points liés au tableaux dynamiques.

 [Delphi Highlander Beta Blogging: Generics](#) (Pour Delphi .NET 2.0 uniquement).

