

Pourquoi un paramètre const change-t-il mystérieusement de valeur ?

par [Sébastien Doeraene](#)

Date de publication : 06/01/2007

Dernière mise à jour :

Vous avez implémenté une routine ou méthode avec un paramètre (appelé Param) déclaré const, et vous remarquez en pas-à-pas la chose suivante. Un appel à une autre routine/méthode, sans même passer Param en paramètre, modifie Param ! Voici l'explication rationnelle du problème (si si, il y en a une).

*Cette « brève » ne concerne que Delphi **Win32**. DotNET et son code managé empêchent ce type de surprises de survenir.*

- I - Mise en situation du problème
- II - Bref rappel sur les paramètres const
- III - L'explication du problème
- IV - Cas de figure posant problème
 - IV-A - Des paramètres var et const pointant au même endroit
 - IV-B - Variable d'instance
 - IV-C - Modification par sous-routine interposée
 - IV-D - Thread concurrent
- V - Mais en pratique, ça arrive vraiment ?
- VI - Comment éviter, ou résoudre, le problème ?

I - Mise en situation du problème

Voici la situation : vous avez implémenté une routine ou méthode avec un paramètre (appelé **Param**) déclaré **const**, et vous remarquez en pas-à-pas la chose suivante. Un appel à une autre routine/méthode, sans même passer **Param** en paramètre, modifie **Param** !

Voici l'explication rationnelle du problème (si si, il y en a une).

II - Bref rappel sur les paramètres const

Avant de commencer, il est indispensable de mettre bien au clair la nature des paramètres **const**. En effet, c'est en raison de cette nature quelque peu particulière que se déclenche le "problème".

Prenons en exemple la routine ci-dessous :

```
// TPoint est déclaré en Windows.pas
procedure WritePoint(Point : TPoint);
begin
  WriteLn(Point.X, ' ', Point.Y);
end;
```

Lors d'un appel à cette routine, le compilateur Delphi duplique la valeur du paramètre effectif dans un emplacement dédié, puis passe l'adresse de cet emplacement à la routine **WritePoint**. Pour un appel comme ça avec un point, qui ne contient jamais que 8 octets, ça n'est pas bien grave. Mais si l'on doit appeler 100 fois des routines en passant en paramètre des **record** de plusieurs dizaines d'octets, cela devient vite gênant.

C'est pourquoi le langage Pascal Objet propose les paramètres de type **const**, qui permettent d'optimiser les appels. Modifions l'en-tête de **WritePoint** pour utiliser un paramètre **const** :

```
procedure WritePoint(const Point : TPoint);
```

Du point de vue de l'intérieur de la méthode, le compilateur va générer des erreurs sur l'utilisation de **Point** de la même manière, et pour les mêmes raisons, que pour des utilisations litigieuses de constantes tout à fait classique, à savoir : affectation à une constante, et passage en paramètre **var/out** d'une constante.

Cela permet d'être sûr qu'un appel à la méthode **WritePoint** ne modifiera pas **Point**, et par conséquent, on n'a plus besoin de dupliquer la valeur du paramètre effectif. On peut envoyer directement l'adresse originale.

Cela fait évidemment gagner énormément de temps. C'est pourquoi on privilégiera l'utilisation de paramètres **const** pour tous les paramètres de types **chaînes**, **tableaux** ou **record**, qui sont les seuls types à s'étendre sur un nombre variable (et donc potentiellement grand) d'octets.

III - L'explication du problème

L'explication est toute proche, maintenant que nous savons exactement comment est transmis un paramètre **const**. Il faut s'intéresser non pas au dedans de la routine, mais bien au dehors.

Lorsque j'appelle **WritePoint**, je lui transmets donc *l'adresse* d'un **TPoint**. Et c'est à cette adresse - la même pour toute la durée de l'exécution de la routine - qu'est lue la valeur du paramètre.

Donc si, de quelque façon que ce soit, durant l'exécution de **WritePoint**, le contenu de la variable originale *change*, la valeur du paramètre changera en conséquence !

Mais comment diable cette variable pourrait-elle changer ? Nous allons voir dans la suite quatre cas de figure, dont trois avec le code.

IV - Cas de figure posant problème

IV-A - Des paramètres var et const pointant au même endroit

```

procedure AddPoints(var Point1 : TPoint; const Point2 : TPoint);
begin
  inc(Point1.X, Point2.X);
  inc(Point1.Y, Point2.Y);
  WriteLn('J''ai avancé de ', Point2.X, ' et monté de ', Point2.Y);
end;

procedure DoublePoint;
var MonPoint : TPoint;
begin
  ReadLn(MonPoint.X);
  ReadLn(MonPoint.Y);
  AddPoints(MonPoint, MonPoint);
  WriteLn('Résultat : ', MonPoint.X, ', ', MonPoint.Y);
end;

```

Dans le morceau de code précédent, la méthode **AddPoints** a pour but d'augmenter **Point1** de la valeur de **Point2**. Nous ne remettons pas en cause le résultat obtenu, mais bien l'affichage de la valeur de **Point2** après l'augmentation.

La procédure **DoublePoint** appelle **AddPoints** pour doubler la valeur d'un point. Et ce en transmettant le même point comme paramètre **var** et comme paramètre **const**. Or le paramètre **var** permet à **AddPoints** de modifier **MonPoint**, et donc **Point2**, qui contient aussi l'adresse de **MonPoint**, en ressent les effets.

IV-B - Variable d'instance

Voici un autre cas de figure, moins direct, qui pose ce problème avec une variable d'instance.

```

type
  TMaClasse = class
  private
    FMonPoint : TPoint;
  public
    procedure DoSomething(const Point : TPoint);
    procedure AppelInitial;
  end;

implementation

procedure TMaClasse.DoSomething(const Point : TPoint);
begin
  WriteLn(Point.X, ', ', Point.Y);
  inc(FMonPoint.Y, 5);
  WriteLn(Point.X, ', ', Point.Y);
end;

procedure TMaClasse.AppelInitial;
begin
  DoSomething(FMonPoint);
end;

```

Dans ce code, si l'on appelle **AppelInitial**, on passe **FMonPoint** par référence à **DoSomething**, qui, en toute apparence, affiche deux fois la même information.

Mais non ! La deuxième fois, le Y a augmenté de 5. Car la variable initiale - **FMonPoint** -, dont on a la référence au travers du paramètre **const**, a été modifiée !

IV-C - Modification par sous-routine interposée

On peut modifier l'exemple précédent pour montrer un exemple de modification indirecte également. Cela n'a rien de différent en soi, mais c'est plus vicieux : on voit moins facilement l'endroit qui coince.

Si on ajoute une méthode **Bouge** dans **TMaClasse**, qui modifie **FMonPoint**, et que l'on appelle cette méthode dans **DoSomething**, on observe le même problème :

```
procedure TMaClasse.Bouge(X, Y : integer);
begin
  inc(FMonPoint.X, X);
  inc(FMonPoint.Y, Y);
end;

procedure TMaClasse.DoSomething(const Point : TPoint);
begin
  WriteLn(Point.X, ' ', Point.Y);
  Bouge(5, -10);
  WriteLn(Point.X, ' ', Point.Y);
end;
```

IV-D - Thread concurrent

Cet exemple demandant trop de code, je ne le reproduirai pas. Mais pensez simplement à un thread concurrent avec l'appel de la routine au paramètre **const**. Si ce thread modifie la variable initiale pendant l'exécution de la routine, le paramètre **const** sera modifié également.

V - Mais en pratique, ça arrive vraiment ?

Ce genre de cas a l'air très improbable comme ça, mais dans certaines applications complexes travaillant sur des **record** dans presque toutes les méthodes, avec donc des paramètres **const** partout pour optimiser, et des appels de *call-back* dans tous les sens, on peut y arriver.

L'exemple concret dans lequel je me suis moi-même retrouvé est malheureusement beaucoup trop complexe pour être expliqué ici. Mais vous pouvez savoir qu'il s'agit d'un cas de mon troisième exemple, à savoir la modification d'une variable d'instance dans une sous-méthode.

Comble de malheur, il s'agissait d'ailleurs d'un appel indirectement récursif à cette sous-méthode.

VI - Comment éviter, ou résoudre, le problème ?

Mais alors, comment l'éviter ? Ou, si vous vous retrouvez dans la situation, comment contourner le problème ?

Il suffit de s'assurer qu'une routine/méthode à laquelle vous passez un paramètre **const** n'a aucun moyen d'obtenir d'une autre façon l'adresse de la variable transmise. Cela peut être fait notamment en copiant le contenu de la variable dans une variable locale, laquelle sera ensuite transmise.

Bien entendu, il ne faut faire cela que dans les cas litigieux que nous venons d'expliquer. Si vous copiez systématiquement tout paramètre **const** avant de le transmettre, vous perdez tout le bénéfice de ce système (éviter la copie justement).