

Guide du débogueur sous Delphi

par Pierre RODRIGUEZ (Pedro) ([Pages de Pedro](#))

Date de publication : octobre 2007

Dernière mise à jour :

Ce tutoriel s'adresse aux débutants qui veulent découvrir les fonctions de débogage sous Delphi.

Ces fonctionnalités sont indispensables à maîtriser pour pouvoir corriger son code.

- I - Introduction
- II - Les points d'arrêts ou breakpoints
 - II.1 - L'ajout simple d'un point d'arrêt
 - II.2 - Les paramètres à prendre en compte
 - II.3 - Les points d'arrêt conditionnels
 - II.3.a - Condition
 - II.3.b - Passes
 - II.3.c - Groupes
 - II.4 - Vérifier ses points d'arrêt
- III - Parcourir le code pas à pas
 - III.1 - Pas à pas (F8)
 - III.2 - Pas à pas approfondi (F7)
 - III.3 - Exécuter jusqu'au curseur (F4)
 - III.4 - Reprendre l'exécution (F9)
- IV - Vérifier ses valeurs
 - IV.1 - La vérification dans l'éditeur de code
 - IV.2 - Les variables locales
 - IV.3 - Les points de suivi
 - IV.4 - Evaluer et modifier les valeurs
- V - Réinitialiser le programme
- VI - Conclusion
- VII - Remerciements


I - Introduction


Rien de plus pénible quand on débute de ne pas arriver à trouver une erreur dans son algorithme ou bien tout simplement identifier la portion de code qui provoque une erreur lors de l'exécution.

Pourtant, dans Delphi, tout existe pour pouvoir retrouver rapidement ses erreurs: le débogueur intégré.

Cet outil est très puissant et permet de vérifier toute sorte de chose lors de l'exécution de son programme.

Ce guide vous permettra de découvrir les principales fonctionnalités disponibles dans Delphi.

 *La version de Delphi utilisée pour ce guide est **BDS 2006** édition Architecte. Il est possible que suivant la version que vous utilisez, les fonctionnalités diffèrent quelque peu.*

 *J'ai gardé par habitude la colorisation syntaxique de Turbo Pascal 7.0. Ne vous étonnez donc pas lorsque vous verrez les captures d'écran.*

II - Les points d'arrêts ou breakpoints

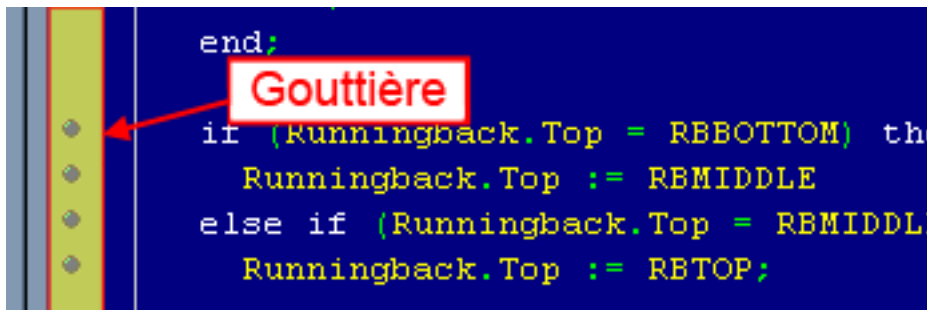
C'est la notion de base à connaître pour pouvoir déboguer.

Il est, en effet, possible de stopper le déroulement d'un programme en définissant des points d'arrêt dans le code.

II.1 - L'ajout simple d'un point d'arrêt

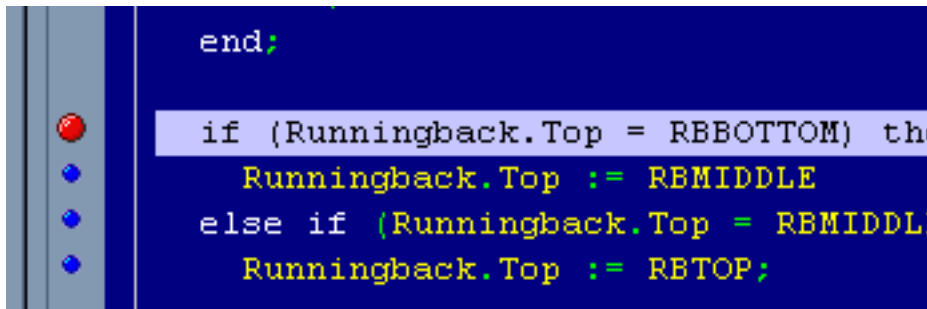
Pour cela, il existe plusieurs méthodes:

- Appui de F5 lorsque le curseur dans l'éditeur de code est sur la ligne
- Clic dans la gouttière à gauche au niveau de la ligne



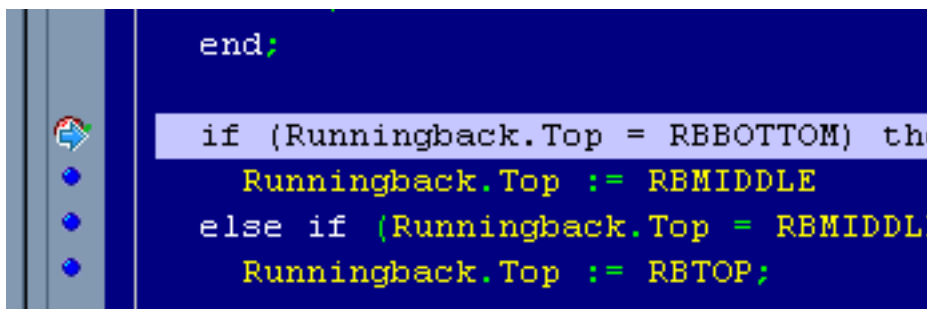
Clic dans la gouttière

Cela aura pour effet de mettre en surbrillance la ligne sélectionnée.



Point d'arrêt

Si vous exécutez le programme, le déroulement sera mis en pause juste avant l'exécution de la ligne du point d'arrêt.




Exécution stoppée

Vous pouvez désactiver le point d'arrêt en cliquant avec le bouton droit sur le rond rouge et en cliquant sur *Activé* dans le menu contextuel. Pour le réactiver, refaites la même manipulation.

Pour supprimer un point d'arrêt, cliquez simplement sur le rond rouge ou bien tapez F5 sur la ligne.

A partir de là, vous avez la possibilité de vérifier, modifier les différentes valeurs de vos variables afin de vérifier votre code.

 *Pour afficher la gouttière de gauche, allez dans le menu Outils\Options.... Sélectionnez ensuite dans l'arborescence de gauche Options de l'éditeur\Affichage puis cochez Gouttière visible. A note qu'il est tout de même possible de mettre un point d'arrêt sans afficher cette gouttière en cliquant juste au bord gauche du code mais l'affichage est beaucoup moins clair à mon goût.*

II.2 - Les paramètres à prendre en compte

On ne peut pas poser des points d'arrêt partout. En effet, certaines lignes de code ne sont pas utilisables. C'est le cas par exemple de certaines lignes sur un **with** telles que celle-ci :

```
with MaClasse do
```

Cette ligne n'appelle aucune méthode. c'est plus une aide au développement qu'une instruction réelle. Vous ne pourrez donc pas poser un point d'arrêt dessus.

Par contre, si le **with** appelle une méthode ou une procédure par exemple:

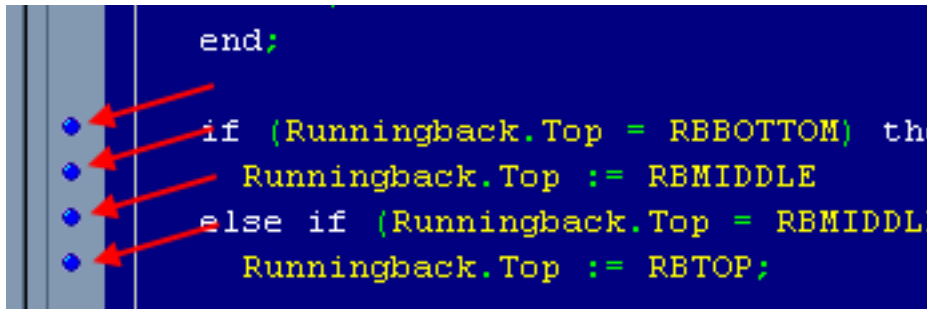
```
with TMaClasse.Create do
```

vous pourrez poser un point d'arrêt: cette ligne appelle le constructeur de TMaClasse.

C'est le cas également des lignes **end;** de fermeture de bloc. On ne peut y poser de point d'arrêt sauf sur la dernière ligne d'une procédure ou d'une fonction.

Heureusement, Delphi vous informe sur les lignes qui peuvent être marquées:

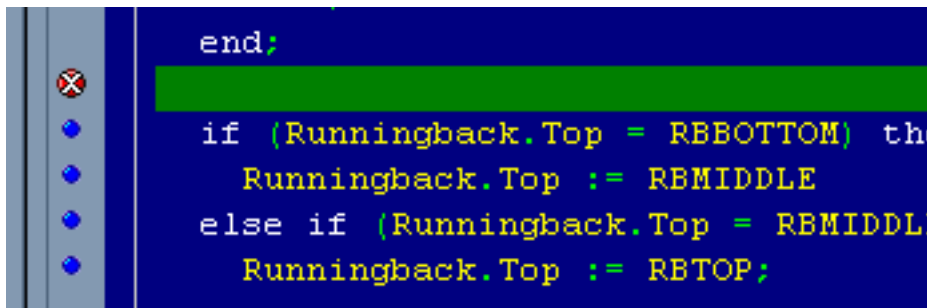
si vous avez bien observé cette fameuse gouttière, vous avez vu des petits points bleus.



Ronds bleus

Ces petits points d'apparence anodine nous indique -entre autres- les lignes utilisables: s'il y a un point bleu, un point d'arrêt peut être posé, sinon, le point d'arrêt n'aura aucun effet. C'est aussi simple que ça.

Si toutefois, vous avez posé un point d'arrêt sur une ligne inutilisable, Delphi vous prévient en colorisant le point d'arrêt différemment lorsque vous exécutez:



Mauvais point d'arrêt

Vous remarquerez également que l'icone dans la gouttière a changé.

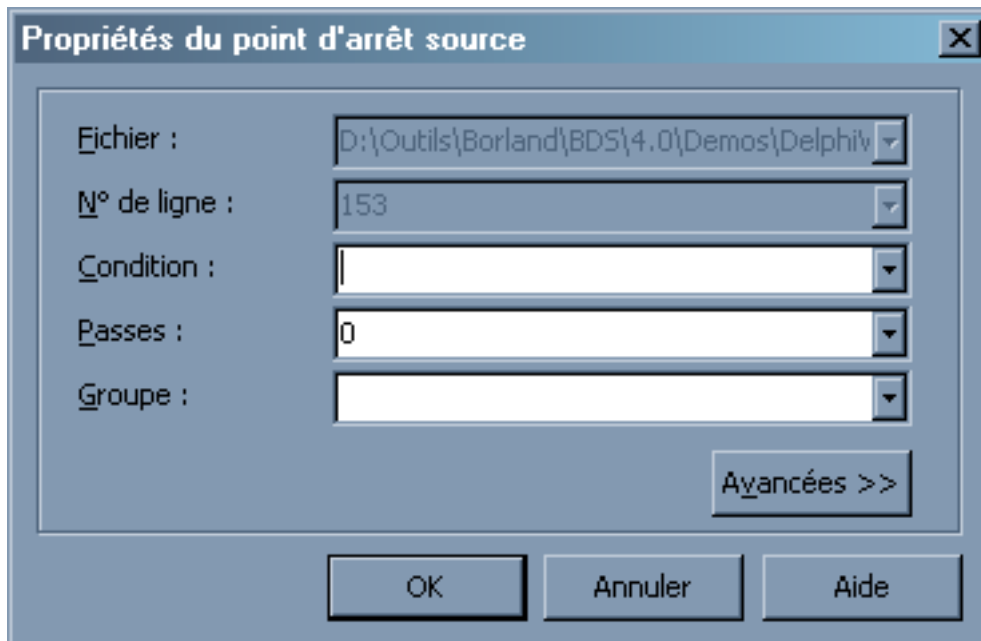
Dans cet exemple, le point d'arrêt a été posé sur une ligne vide. Il n'y a pas de point bleu.

Cette colorisation se fait au moment de l'exécution et seulement à ce moment-là.

II.3 - Les points d'arrêt conditionnels

Il est également possible de définir des points d'arrêt qui stoppe le déroulement du programme suivant une ou plusieurs conditions.

Pour cela, ajoutez un point d'arrêt comme nous l'avons vu plus haut. Cliquez avec le bouton droit sur le rond rouge dans la gouttière puis cliquez sur *Propriété du point d'arrêt* dans le menu contextuel qui apparait. Cette boîte de dialogue apparait:



Propriété du point d'arrêt

Voici la description des champs disponibles:

- Condition: Définit la condition à laquelle le point d'arrêt doit stopper le déroulement du programme
- Passes: Définit le nombre de fois où le programme passe sur le point d'arrêt sans s'arrêter.
- Groupe: Crée un groupe de points d'arrêt auquel appartient ce point d'arrêt.
- Bouton *Avancées*: Permet d'accéder à des paramètres étendus qui vont modifier l'action qui se produit lorsque le point d'arrêt se déclenche.

II.3.a - Condition

Saisissez la condition comme si vous la saisissiez dans l'éditeur de code. Vous pouvez définir plusieurs conditions de la même façon que dans le code en utilisant les opérateurs booléens (or, and, xor, ...).

Exemple de condition multiple:

```
(i=10) and (j=2)
```

Ce paramètre stoppera le déroulement du programme lorsque les valeurs de i et de j seront respectivement 10 et 2. Autrement, le point d'arrêt ne se déclenche pas.

II.3.b - Passes

Définissez grâce à ce paramètre le nombre de fois où le point d'arrêt est survolé avant qu'il ne stoppe le programme. A noter que si vous définissez un nombre n, l'exécution sera stoppée juste avant la énième passe. Par exemple sur le code suivant:

```
var
```


```
I: Integer;  
begin  
  for I := 1 to 10 do  
    ShowMessage(IntToStr(I));  
end;
```

Si vous marquez un point d'arrêt sur la ligne du ShowMessage et que vous mettiez 10 au nombre de passes, le programme stoppera à la 10ème passe. C'est à dire, dans le cas de notre exemple juste avant qu'il n'affiche la boîte de dialogue "10".

II.3.c - Groupes

Ce paramètre permet de définir des groupes de points d'arrêt. Si vous saisissez un nom dans cette liste, le nom que vous avez saisi sera accessible pour tous les autres points d'arrêt.

Ainsi, lorsque vous modifiez une donnée, la modification est répercutée sur tous les points d'arrêt du même groupe.

 *Si vous enlevez le point d'arrêt, vous perdez les paramètres que vous avez définis. Toutefois, ils restent disponibles dans les listes déroulantes de chacun des champs dans la boîte de dialogue.*

II.4 - Vérifier ses points d'arrêt

Vous pouvez avoir une vue d'ensemble des points d'arrêt définis en affichant la liste de ces points.

Pour cela, allez dans le menu *Voir\Fenêtres de débogage\Points d'arrêt*.


Une liste exhaustive des points spécifiés apparaît ainsi que leurs paramètres.

III - Parcourir le code pas à pas

Une fois le déroulement stoppé, vous pouvez continuer le déroulement pas à pas (ligne par ligne) pour identifier précisément la ligne de code fautive. Il existe plusieurs façons de parcourir le code une fois le programme arrêté. Nous verrons ici les 4 plus utilisées.

III.1 - Pas à pas (F8)

L'appui de F8 ou bien sur l'icone correspondant dans la barre d'outils passera le déroulement à la ligne suivante dans l'éditeur de code en restant dans la même méthode.

 *Attention, il faut bien comprendre que la ligne mise en surbrillance **n'est pas encore** exécutée.*

Autrement dit, si la ligne en question est une affectation d'une variable, l'affectation n'est pas encore effectuée.

III.2 - Pas à pas approfondi (F7)

L'appui de F7 produit la même action que l'appui sur F8 à la grosse différence près que le déroulement "rentre" dans les fonctions citées.

Prenons l'exemple suivant:

```
function Test: integer
begin
  Result := 10;
end;

procedure Execute;
var i: integer;
begin
  i := Test;
end;
```

Si l'on pose un point d'arrêt sur la ligne

```
i := Test;
```

le déroulement s'arrête sur cette ligne. Si vous appuyez sur F8, la ligne courante passe au dessous. Par contre, si vous appuyez sur F7, la ligne passe dans la fonction Test.

Pour déboguer efficacement, il faut apprendre à utiliser ces deux fonctionnalités et jongler entre elles.

III.3 - Exécuter jusqu'au curseur (F4)

Vous pouvez indiquer au débogueur de stopper le déroulement de l'exécution sur la ligne du curseur dans l'éditeur de code. Autrement dit, sans ajouter de point d'arrêt. Très pratique pour tester rapidement le code que l'on vient d'écrire.

III.4 - Reprendre l'exécution (F9)

Si vous voulez reprendre le déroulement normal du programme, appuyez sur F9 ou bien cliquez dans le menu *Exécuter**Exécuter* ou sur le bouton *ad hoc* dans la barre d'outils ou.

Par contre, cela ne désactive pas les différents points d'arrêt définis qui pourront toujours se déclencher.

IV - Vérifier ses valeurs

Maintenant que la notion des points d'arrêt est digérée, passons au vif du sujet.

Car c'est bien beau de faire stopper le programme sur une ligne donnée mais il faudrait pouvoir contrôler nos variables.

Les sections suivantes partiront du principe que le programme est déjà exécuté et est stoppé sur un point d'arrêt.

IV.1 - La vérification dans l'éditeur de code

C'est la plus simple à obtenir. Il suffit pour cela de positionner votre curseur sur la variable qui vous intéresse. Un hint (bulle d'aide) s'affichera alors avec le contenu de la variable survolée.

Cette méthode est intéressante car il n'y a rien à mettre en place. Toutefois, cela devient assez fastidieux lorsque vous devez surveiller plusieurs variables en même temps.

IV.2 - Les variables locales

Cette liste se remplit automatiquement avec les variables déclarées dans la méthode parcourue actuellement et affiche leur contenu en temps réel lors du parcours du code.

Il est impossible de modifier cette liste.

Par défaut, cette boîte se situe juste en dessous de la Liste des points de suivi.

The screenshot shows the Delphi IDE interface. On the right, a code editor displays a procedure named `Procedure TForm1.FormShow(Sender: TObject)`. The code includes a `var` block with `I: Integer;` and `MonType: TMonType;`, followed by a `begin` block containing a `for` loop from 1 to 10. Inside the loop, there is a `ShowMessage` call and a `with MonType do` block containing assignments for `Phrase`, `Entier`, `Booleen`, and an update to `Phrase`. The procedure ends with `end.`

On the left, the 'Variables locales' window is open, showing the current procedure `TForm1.FormShow(???)`. It contains a table with the following data:

Variable	Valeur
	E2171 La variable 'Self' est inaccessible ic...
	E2171 La variable 'Sender' est inaccessibl...
	E2171 La variable 'I' est inaccessible ici d...
MonType	('Ceci est un test', 10, True)

A red box highlights the 'Variables locales' window, and a red arrow points to it from a label 'Variables locales'.

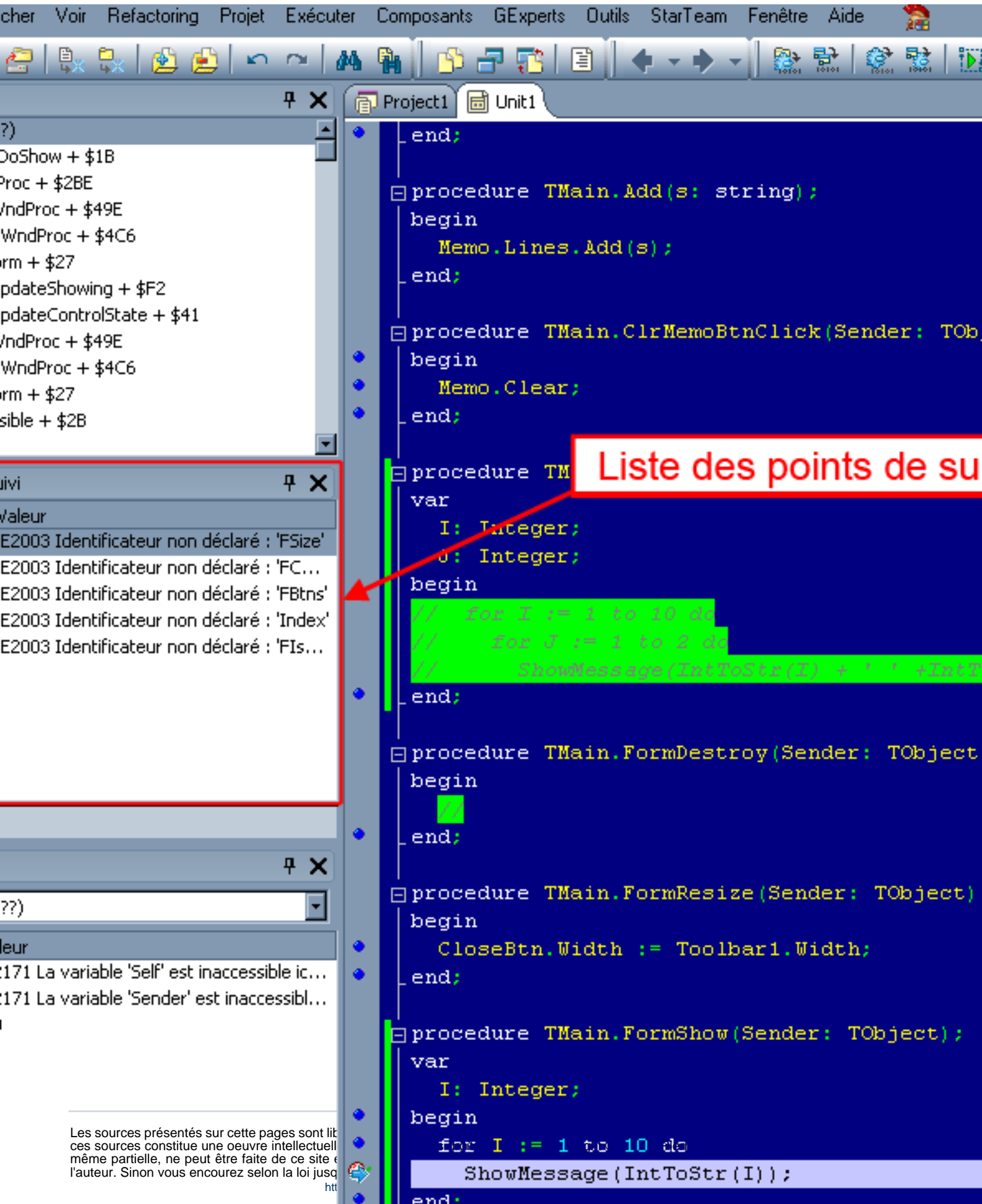
Variables locales

Sinon pour l'afficher, cliquez dans le menu *Voir\Fenêtres de débogage\Variables locales*.

IV.3 - Les points de suivi

Lorsqu'il faut surveiller plusieurs valeurs lors du débogage, il est intéressant de les avoir toutes sous les yeux en même temps. C'est ce que permettent les points de suivi.

Avant toute chose, il faut vérifier que la boîte qui contient les points de suivi est affichée. Si vous avez laissé les options par défaut, elle se trouve dans la partie gauche de l'EDI, au milieu:



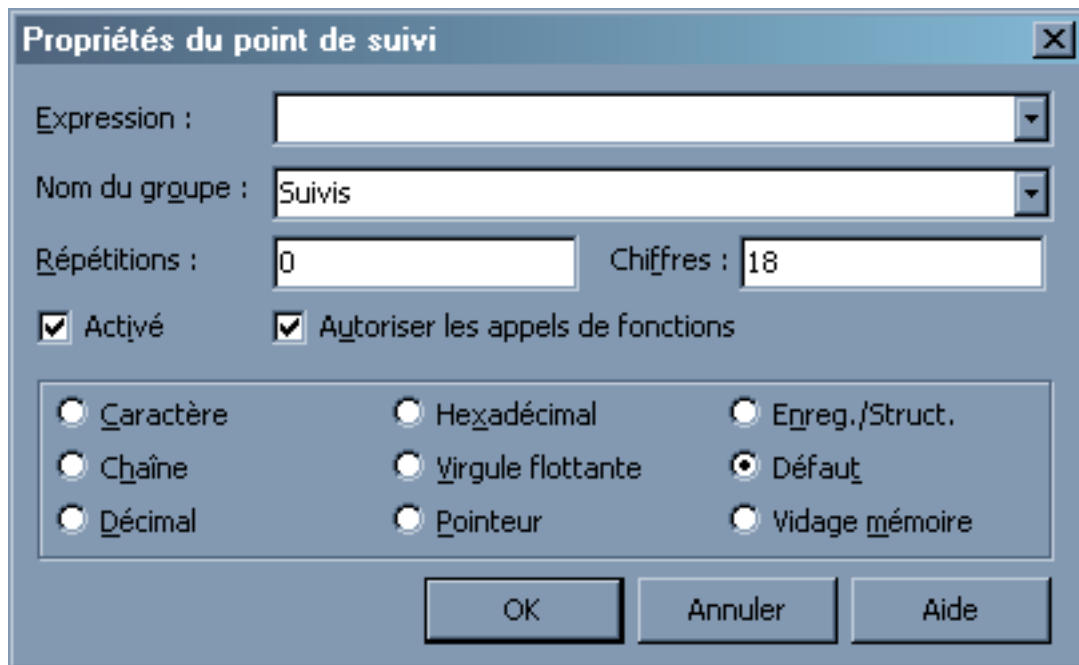
Liste des points de suivi

Sinon, vous pouvez l'afficher en cliquant dans le menu *Voir\Fenêtres de débogage\Points de suivi*.

Cette liste contient tous les points de suivi ainsi que les valeurs courantes.

Pour ajouter un point de suivi, cliquez dans l'éditeur de code sur la variable que vous souhaitez contrôler puis faites Ctrl+F5. Cela aura pour effet d'ajouter cette variable dans la liste.

Vous pouvez également ajouter un point de suivi en faisant *Exécuter\Ajouter un point de suivi...*. La boîte de dialogue suivante apparaît alors:



Ajouter un point de suivi

Saisissez alors dans le champs **Expression** la valeur que vous souhaitez surveiller.

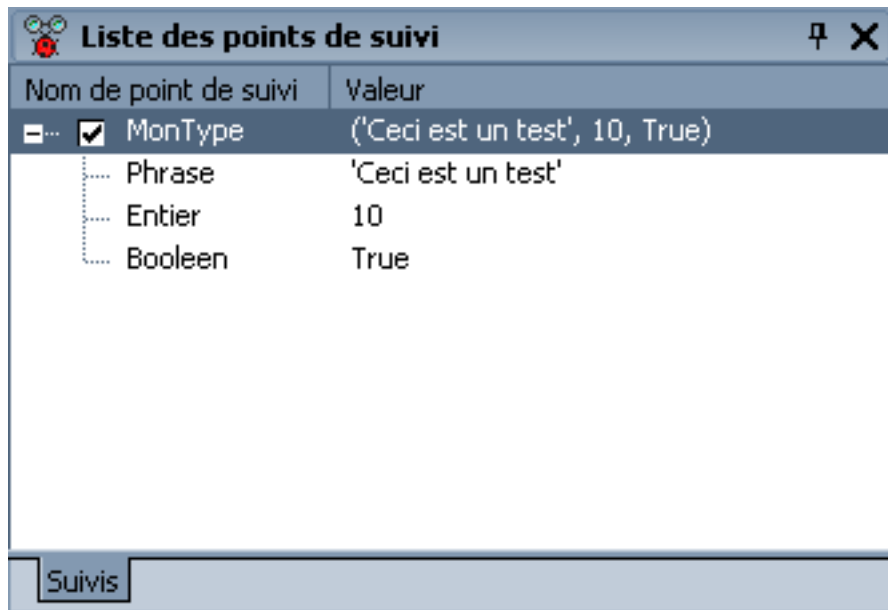
Si vous utilisez une fonction dans votre valeur, par exemple:

```
Length(MaPhrase)
```

vous devez cocher **Autoriser les appels de fonction**.

En règle générale, vous pouvez laisser cette option cochée quelle que soit la valeur.

On peut noter également que lorsque l'on suit un type structuré (un record ou une classe), on peut accéder à tous les champs de cette structure:



Suivi d'une type structuré

Toutefois, il y a des situations où la liste des points de suivi sera incapable de vous afficher la valeur. Cela peut provenir de plusieurs raisons:

- La variable a été supprimée par le compilateur: La variable n'est pas ou plus utilisée à l'endroit du suivi, il n'y a plus aucun accès à cette variable
- La variable n'existe pas: La variable n'existe pas dans la portion de code parcourue.
- Violation d'accès: La variable (en général, une classe) n'a pas été créée.
- etc.

La liste des points de suivi opère sur les variables telles qu'elles sont définies dans le code. C'est à dire que par exemple, une variable locale dans une procédure ne sera vérifiable que dans cette procédure sauf si l'on parcourt une autre portion de code qui contient cette variable.

Par exemple, reprenons le code précédent:

```

function Test: integer
begin
    Result := 10;
end;

procedure Execute;
var i: integer;
begin
    i := Test;
end;
    
```

et ajoutons un suivi sur **Result**. Comme vous pouvez le voir, cette variable est seulement disponible dans la fonction Test. Si l'on met un point d'arrêt sur la ligne

```

i := Test;
    
```

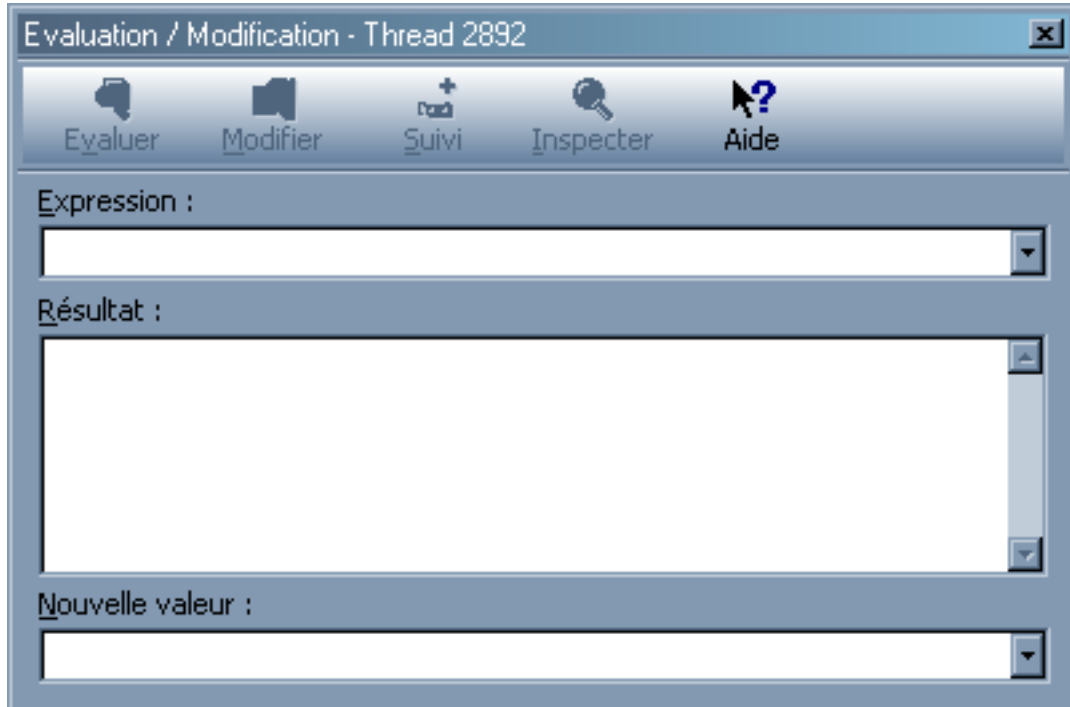
la variable **Result** ne sera pas accessible. Elle le sera seulement lorsque le déroulement passera dans cette fonction Test.

Un clic droit sur un des éléments de la liste donne accès à des actions telles que:

- Modifier le point de suivi
- Ajouter un point de suivi...
- Désactiver le point de suivi
- Supprimer le point de suivi
- Copier la valeur du suivi
- Copier le nom du suivi
- Activer tous les points de suivi
- Désactiver tous les points de suivi
- Supprimer tous les points de suivi
- etc.

IV.4 - Evaluer et modifier les valeurs

Pour compléter toutes ces fonctions, il est également possible de modifier la valeur des variables. En effet, si vous faites *Exécuter/Evaluer/Modifier...* (ou Ctrl+F7), Delphi affichera une boîte de dialogue comme celle-ci:



Evaluation/Modification

Le champs **Expression** permet de spécifier la variable que l'on veut évaluer ou modifier. Le champs **Résultat** est en lecture seule et affiche la valeur contenue dans la variable. Le champs **Nouvelle valeur** permet de modifier la valeur de l'expression.

Pour utiliser cette boîte de dialogue, il faut tout d'abord spécifier la variable. Ensuite, cliquez sur **Evaluer** pour remplir le champ **Résultat**

Si vous spécifiez une nouvelle valeur dans le champs du même nom, cliquez ensuite sur **Modifier**. Si tout s'est bien passé, vous verrez le résultat se modifier et prendre la valeur que vous avez spécifiée.

Ainsi, si vous poursuivez le déroulement de l'exécution, ce sera avec la nouvelle valeur que vous avez définie.

Il est également possible d'ajouter un point de suivi depuis cette boîte de dialogue en cliquant sur **Suivi**. Un point de suivi s'ajoutera automatiquement alors dans la liste des points de suivi.

Avant BDS2005, cette méthode était la plus rapide puisque le système d'affichage des valeurs dans les bulle d'aide (Hint) n'existait pas encore.

V - Réinitialiser le programme

Vous pouvez également stopper net le débogage du programme et le programme lui-même en cliquant dans le menu *Exécuter**Réinitialiser le programme* ou en faisant Ctrl+F2.

De cette façon, le programme s'arrête et vous revenez dans l'éditeur de code en mode conception.

C'est très pratique lorsque, en plein débogage, vous identifiez une ligne fautive et que vous voulez la corriger tout de suite. Vous n'avez pas besoin de fermer votre programme manuellement, ce qui se révèle indispensable lorsque votre programme provoque des erreurs ou des boucles interminables par exemple.

VI - Conclusion

Maintenant que vous maîtrisez toutes ces fonctionnalités, vous pourrez déboguer facilement vos programmes et contrôler parfaitement son déroulement.

Pour aller plus loin encore dans le débogage, je vous invite à consulter le lien suivant:  <http://www.blong.com/Conferences/DCon2000/Debugging/Debugging.htm> qui explique -entre autres- l'utilisation de la fenêtre CPU.

VII - Remerciements

Je tiens à remercier **Sébastien Doeraene (sjrd)** pour sa relecture et ses conseils avisés.

 [Retour à l'accueil](#)

