Utilisation simplifiée des objets automation

Auteur : Michaël Moreno
Web : http://michael.moreno.free.fr/

Niveau de programmation : Avancé

NB : La programmation des objets automation requiert un bon niveau de programmation. Une mauvaise utilisation de ces objets ou une erreur de la part du programmeur peut créer une grande instabilité du système d'exploitation et une perte irrémédiable des données présentes sur le disque dur. Les programmeurs non expérimentés et ne maîtrisant pas Delphi, VBA pour Excel et la base de registre ne devraient pas étudier ce tutoriel.

L'auteur ne saurait être tenu responsable des dégradations de toute sorte se produisant sur n'importe quel ordinateur suite à la lecture et à l'utilisation des techniques présentées dans ce document.

Requis (il me semble) : Excel 97, Delphi 3 ou supérieur, IE 4, win 95.
Idéal : Excel 97 - 2000, Delphi 5 ou 6, win XP, NT ou 2000, IE 5 ou supérieur.

Attention : lors d'un débuggage sous win 95, 98 ou Me l'interruption avec Ctrl-F2 fera planter votre machine systématiquement. L'utilisation de Ctrl-F7 sur un tableau de variant ou l'un de ses éléments aussi… Et la liste des problèmes de ce type est très longue.

Aucune aide supplémentaire ne sera fournie par l'auteur. Inutile donc de me contacter pour savoir comment faire telle ou telle chose comme par exemple créer un serveur automation d'accès à une base de données. Le site http://www.developpez.com/ ou le forum nzn.fr.delphi du newsgroup news.vienneinfo.org devraient vous apporter pratiquement toutes les solutions à vos problèmes.

Les DLL (*Dynamic Linked Library*) contiennent des fonctions essentielles appelés par les logiciels. La simplicité de leur déploiement et leur réutilisabilité au sein de plusieurs projets rendent ces bibliothèques de fonctions et d'objets très intéressantes pour les professionnels. Elles présentent pourtant certaines limites. Les DLL doivent être totalement indépendantes du langage utilisé. En pratique, on s'aperçoit qu'elles sont au contraire très dépendantes de celui-ci, soit en raison d'une mauvaise programmation soit parce qu'elles ont été conçues pour être utilisées avant tout avec un certain langage de programmation. Ainsi, on dénombre sur le net une myriade de DLL écrite en C++ que l'on ne peut utiliser en pratique qu'avec la dernière version de Visual C++, même les autres compilateurs C++ ne parviennent pas à compiler les fichiers d'en-tête fournis. Alors autant dire que de se servir de ces DLL sous Visual Basic ou Delphi est un véritable challenge.

Il existe pourtant une alternative simple : les objets Automation (extension de COM – *Component Object Model*). La technologie derrière ces objets dépassent de loin le cadre de ce bref tutorial. Toutefois, l'utilisation de ces objets est des plus simples et ce sous n'importe quel langage de programmation « moderne » tels que Basic, C++ ou encore Pascal Objet. La programmation de ces objets est en revanche légèrement plus difficile. Sachant que l'objet ainsi programmé sera utilisable en réseau sous Delphi, C++ Builder, Visual C++, Visual Basic, Excel, Word, ACCESS, etc la difficulté supplémentaire de leur programmation est grandement récompensée.

Le but de ce tutorial est d'introduire la programmation des objets Automation sans aucune autre prétention que celle de créer quelques fonctions simples appelables depuis Excel ou Delphi. Le logiciel de

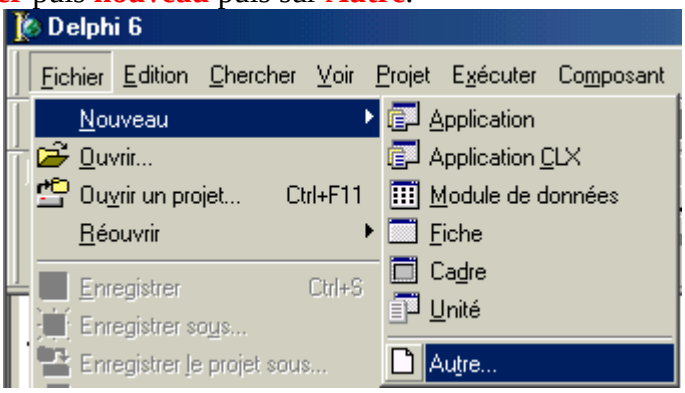
programmation utilisé est Delphi 6 pro. Les utilisateurs de C++ Builder n'auront aucune difficulté à créer leurs propres objets en lisant ce tutorial car le wizard est commun à Delphi et Builder.

I. Programmation de l'objet Automation

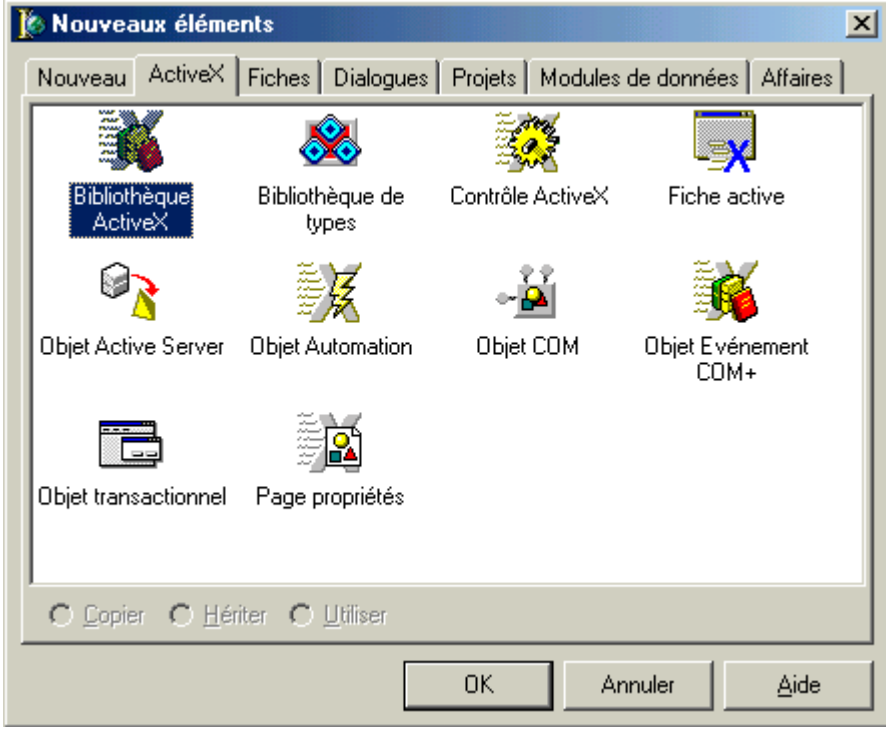
Les objets automation sont contenus dans des bibliothèques de types, elles même faisant partie de bibliothèque ActiveX. Il est donc nécessaire avant de créer un tel objet de créer une bibliothèque ActiveX et une bibliothèque de types.

A. Création de la bibliothèque ActiveX

Cliquez sur le menu **Fichier** puis **nouveau** puis sur **Autre**.



Sur l'onglet **ActiveX**, sélectionnez l'icône de la **bibliothèque ActiveX** et cliquez sur OK.



Il apparaît alors un nouveau projet de type **library** :

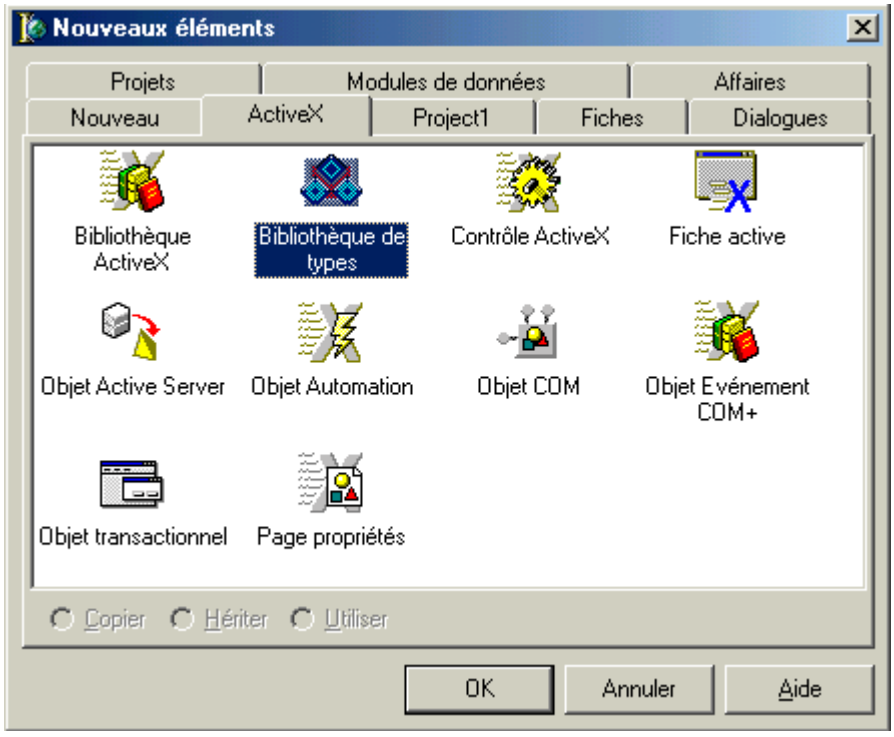
```
library Project1;

uses
  ComServ;
```

```
exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.RES}

begin
end.
```

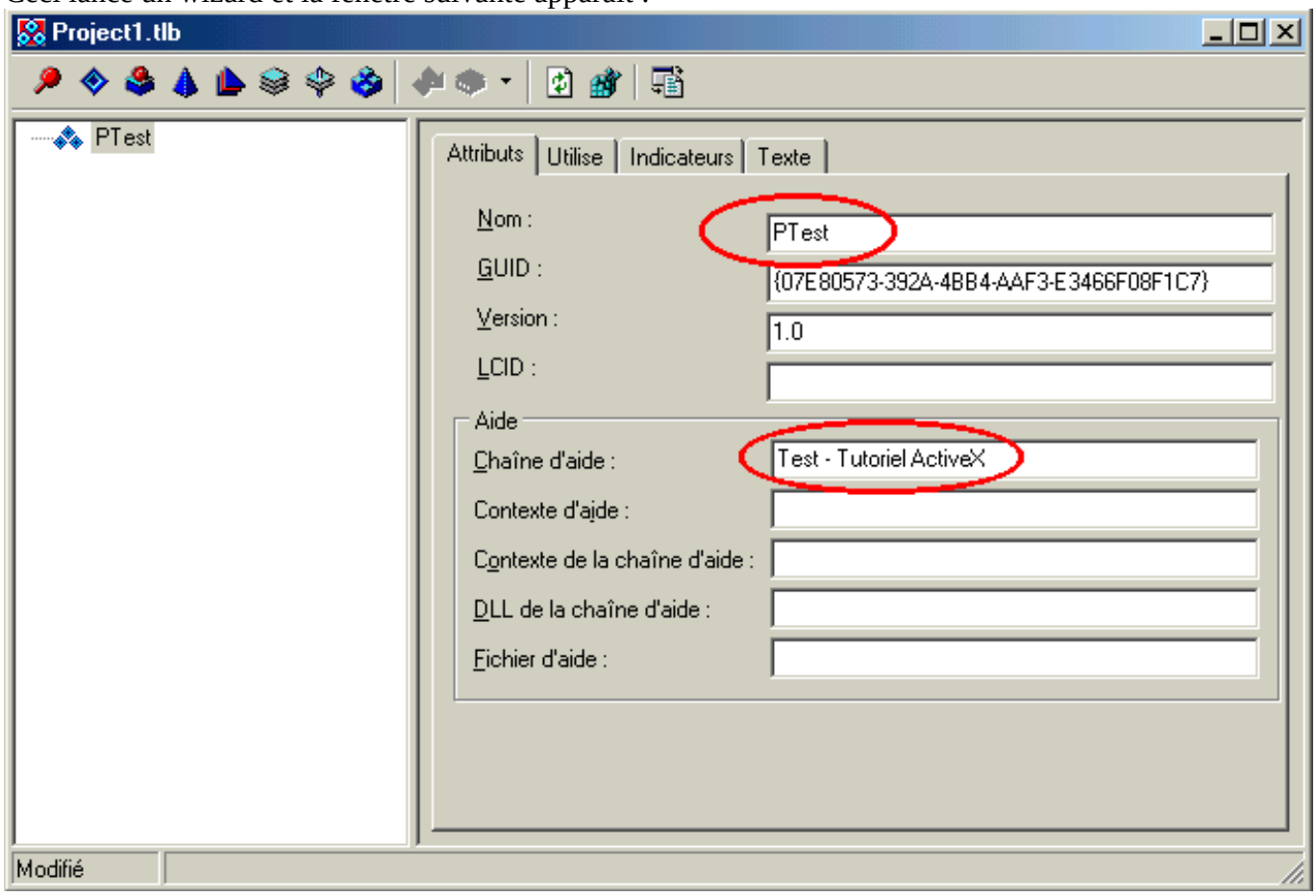
Remarquez ici l'exportation des fonctions DllGetClassObject, DllCanUnloadNow, DllRegisterServer, DllUnregisterServer. Toutes bibliothèques ActiveX doit présenter ces fonctions qui vont permettre le recensement des objets dans la base de registre. Ne vous en préoccupez pas pour l'instant.

B. Création de la bibliothèque de types

Il faut à présent créer la bibliothèque de types. Cliquez sur le menu **Fichier** puis **nouveau** puis sur **Autre**. Sur l'onglet **ActiveX**, sélectionnez l'icône de la **bibliothèque de types** et cliquez sur OK.



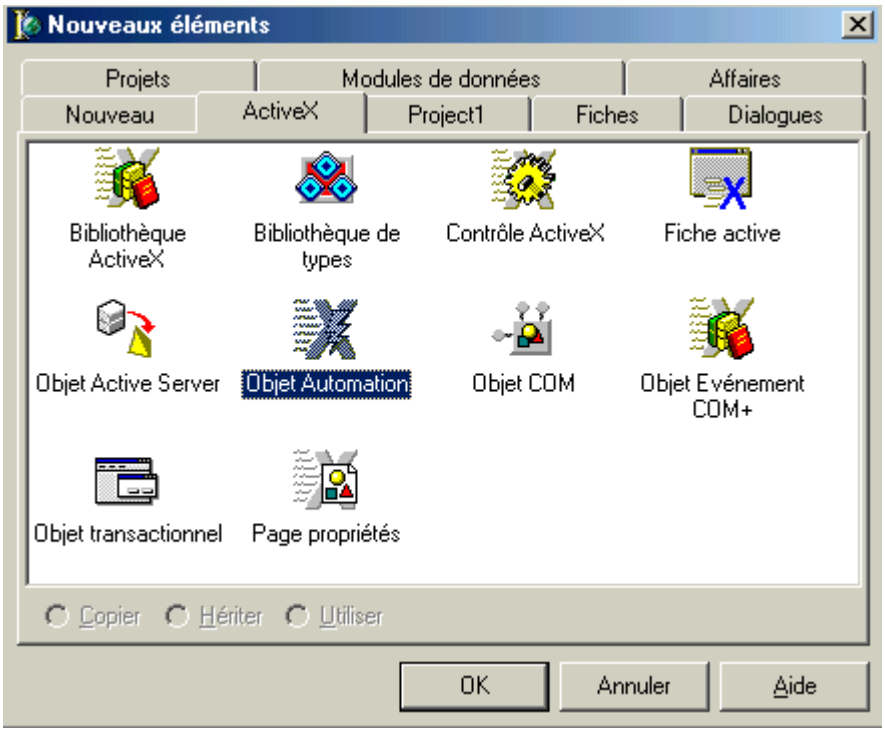
Ceci lance un wizard et la fenêtre suivante apparaît :



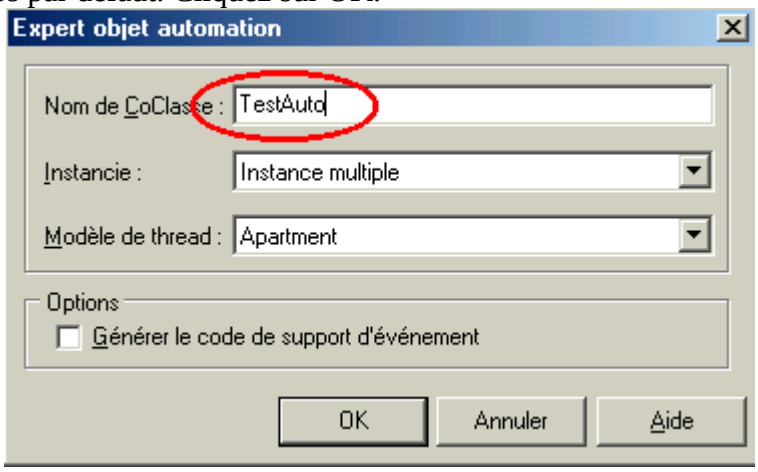
Changez le champ nom en **PTest** et la chaîne d'aide en « **Test – Tutoriel ActiveX** ».

C. Création de l'objet automation

Il faut à présent créer l'**objet Automation**. Cliquez sur le menu **Fichier** puis **nouveau** puis sur **Autre**. Sur l'onglet **ActiveX**, sélectionnez l'icône de l'**objet Automation** et cliquez sur OK.

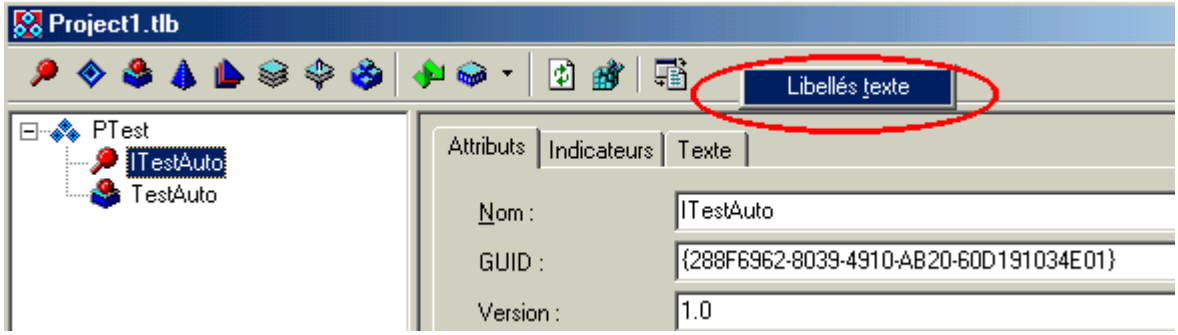


Une fenêtre de renseignements apparaît. Tapez pour nom de Coclass : « **TestAuto** ». Laissez les autres valeurs de champs par défaut. Cliquez sur OK.

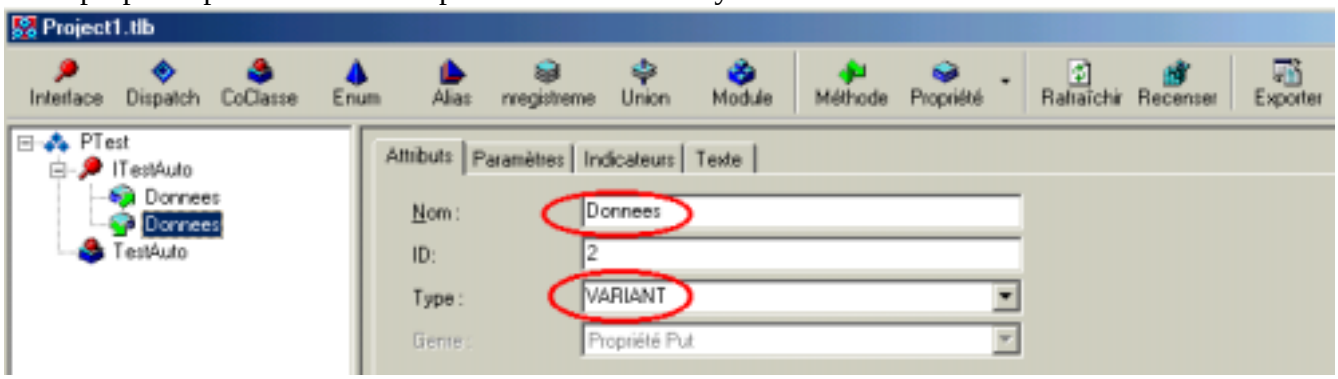


D. Ajout de propriétés

Avant d'ajouter les propriétés, cliquez droit sur la barre des icônes de la fenêtre de la bibliothèque de types et sélectionnez les libellés texte. Cela affichera le texte sous les icônes de la barre des taches.

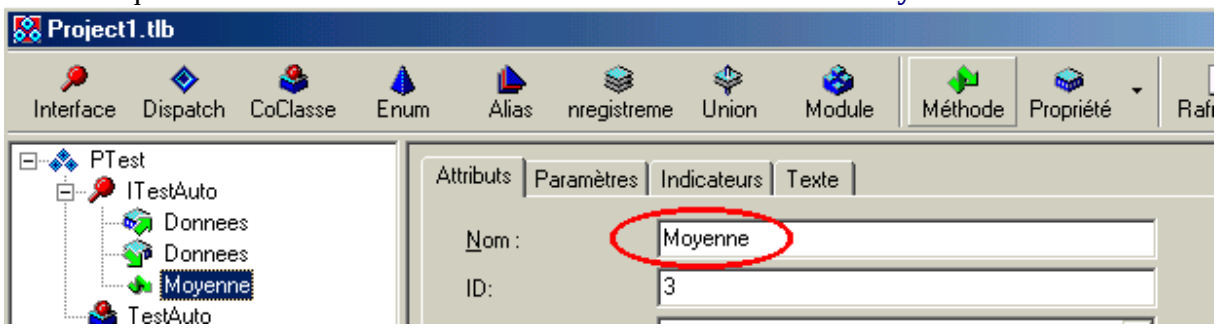


Sélectionner l'interface de l'objet **ITestAuto**[1]. Cliquez sur l'icône « **Propriété** » pour ajouter une nouvelle **property** à votre objet. Donnez lui comme nom **Donnees** et sélectionnez le type **VARIANT**. Cette propriété permettra de faire passer un variant array de double.

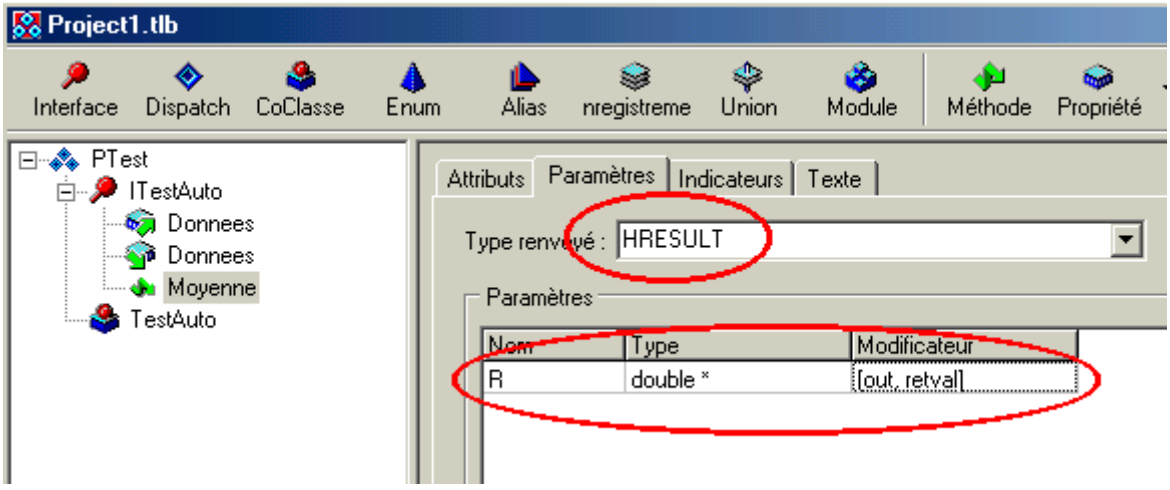


E. Ajout de méthode

A présent, nous allons créer une méthode permettant de calculer la moyenne des données passées dans la **property**. Dans le cadre de ce tutorial, il eut été possible de passer les données en paramètres de la méthode. Cliquez sur l'icône « méthode » et renommez la méthode « **Moyenne** ».



Cette méthode doit renvoyer sous la forme d'un double la valeur moyenne. Dans l'onglet « **Paramètres** », créez une variable de nom R de type « **double *** » (tapez le en toute lettre car ce type - aussi incroyable que cela puisse paraître – n'est pas listé). Double-cliquez sur la cellule du modificateur et sélectionnez la case « **Valeur renvoyée** » ce qui sélectionnera automatiquement la case « **sortie** ». Le type renvoyé HRESULT ne doit pas être modifié.



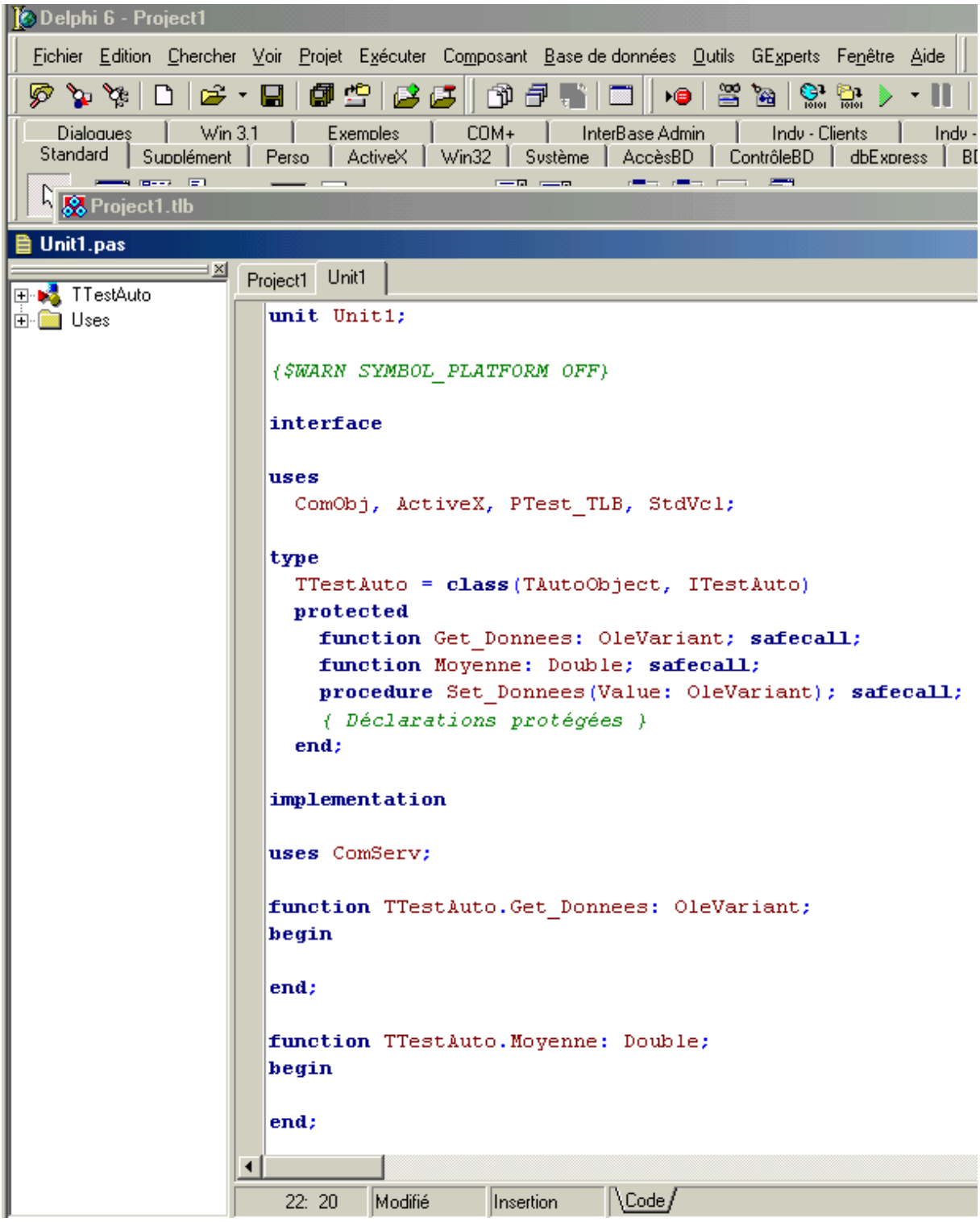
[1] On ajoute les propriétés et les méthodes à l'interface de l'objet. L'objet hérite de cette interface et de l'objet TAutoObject. Le client de l'objet ne connaît pas la coclasse TestAuto et n'accède à l'objet qu'au travers de son interface. Il est possible d'implémenter plusieurs interfaces pour un même objet et de faire hériter les interfaces entre elles.

F. Rafraîchissement

Il est temps de rafraîchir l'implémentation du code. Cliquez sur le bouton « **Rafraîchir** ». Ceci va créer le code de la structure de l'objet dans les fichiers « .pas ».

Parfois cette méthode rafraîchit mal le code. Sous Delphi 6 le nombre de bugs du wizard a été nettement diminué par rapport à Delphi 5. Si votre code est mal rafraîchit, l'enregistrement des fichiers permet souvent de résoudre les problèmes de rafraîchissement. Les messages d'erreur qui peuvent survenir risquent de vous laisser perplexe. Personnellement, l'apprentissage a été long et difficile.

Une fois rafraîchit le code source apparaît :



G. Programmation de l'objet

Toute l'architecture a été créée. Il reste à coder l'objet aussi simplement que n'importe quel objet Pascal à quelques différences près.

1. Rajout de variables private

Dans la partie **private**, ajoutez les variables FDonnees et FNbDonnees comme suit :

```
type
  TTestAuto = class(TAutoObject, ITestAuto)
  private
    FDonnees : OleVariant;
    FNbDonnees : Integer;
  protected
    function Get_Donnees: OleVariant; safecall;
    function Moyenne: Double; safecall;
    procedure Set_Donnees(Value: OleVariant); safecall;
    { Déclarations protégées }
  end;
```

Rajoutez l'unité **Variants** dans la clause des **uses**.

```
interface

uses
  Variants, ComObj, ActiveX, PTest_TLB, StdVcl;

type
  TTestAuto = class(TAutoObject, ITestAuto)
  private
    FDonnees : OleVariant;
```

2. Programmation de Get et Set Donnees

L'objet Automation doit savoir communiquer avec l'extérieur. Les propriétés permettent de simplifier la transmission des paramètres. Dans le cas des objets les property n'existent pas et sont remplacées directement par deux méthodes « set » et « get ». Le code est présenté et commenté ci-dessous :

```
function TTestAuto.Get_Donnees: OleVariant;
begin
  varCopy(result,FDonnees); // copie de FDonnees dans result
end;

procedure TTestAuto.Set_Donnees(Value: OleVariant);
begin
  if varIsArray(Value) then // si le variant est bien un tableau
  begin
    varCopy(FDonnees, value); // copie du variant
    FNbDonnees := varArrayHighBound(Value,1) + 1; // il est supposé que le tableau commence à l'indice 0
    // et que le tableau est un vecteur et non une matrice
  end;
  // else on lève une exception avec raise ...
end;
```

3. Programmation de la méthode Moyenne

Le calcul de la méthode moyenne est des plus simples :

```
function TTestAuto.Moyenne: Double;
var i : integer;
begin
  if FNbDonnees = 0 then
    raise Exception.Create('Pas de données.');

  result := 0;
  try // Ne pas oublier de rajouter l'unité SysUtils dans la liste des uses
    for i:=0 to FNbDonnees-1 do
      result := result + FDonnees[i];
    result := result / FNbDonnees;
  except
    raise Exception.Create('Une erreur est survenue...');
  end;
end;
```

H. Initialisation et destructeur

Il est « interdit » de surcharger le **constructor** de l'objet Automation. L'objet parent TautoObject implémente le **constructor Create** et la dernière méthode appelée par celui-ci est la **procedure Initialize**. C'est cette **procedure** que l'on réintroduit dans notre objet pour initialiser correctement les différentes variables de l'objet. L'implémentation du **destructor Destroy** est en revanche identique à celle d'un objet Pascal usuel.

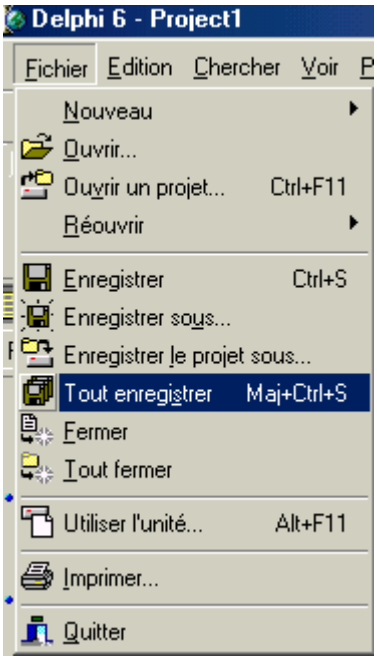
```
type
  TTestAuto = class(TAutoObject, ITestAuto)
  private
    FDonnees : OleVariant;
    FNbDonnees : Integer;
  protected
    function Get_Donnees: OleVariant; safecall;
    function Moyenne: Double; safecall;
    procedure Set_Donnees(Value: OleVariant); safecall;
    { Déclarations protégées }
  public
    procedure Initialize; override;
    destructor Destroy; override;
  end;
```

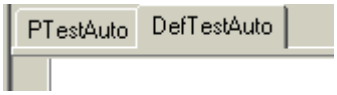
L'implémentation du **destructor** et de **l'initialize** est immédiate :

```
destructor TTestAuto.Destroy;
begin
  varClear(FDonnees); // Libération de la mémoire*
  inherited;
end;

procedure TTestAuto.Initialize;
begin
  inherited;
  FNbDonnees := 0;
end;
```

I. Enregistrement

Suivez la procédure suivante pour enregistrer votre projet et unité pascal.



L'unit1 devient : DefTestAuto et le Project1 devient PTestAuto : 

J. Recensement de l'objet

Afin que les autres logiciels connaissent l'existence des objets Automation, il est nécessaire de les recenser au sein de la base de registre.

Après avoir fini l'exercice, n'oubliez pas de dérecenser votre serveur ActiveX afin de ne pas polluer inutilement votre base de registre.

En fait il est possible de recenser le serveur à l'aide de la ligne de commande du type suivant :

Regsvr32 « Chemin\NomProjet.dll »

Et de le dérecenser avec la ligne de commande similaire à :

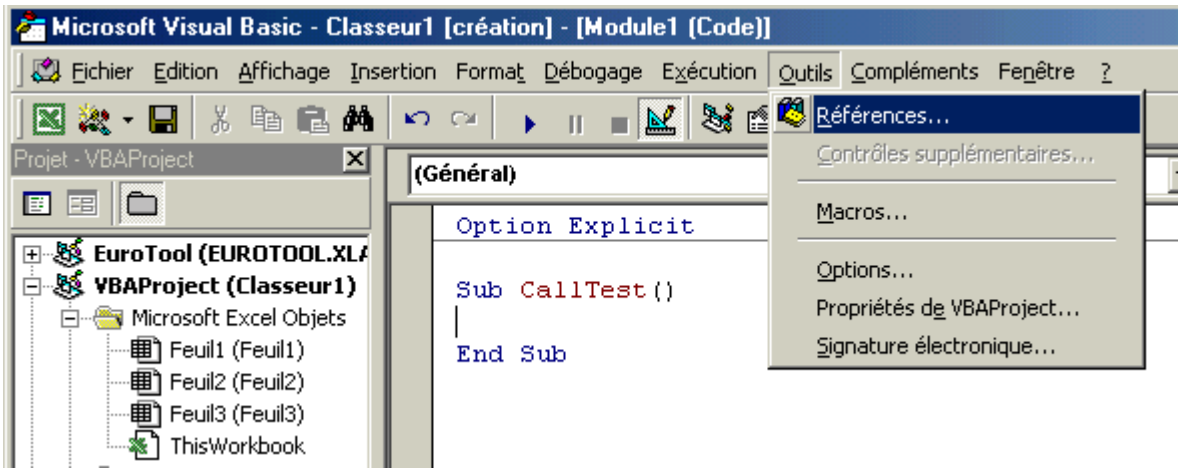
Regsvr32 **-u** « Chemin\NomProjet.dll »

II. Interface avec Excel

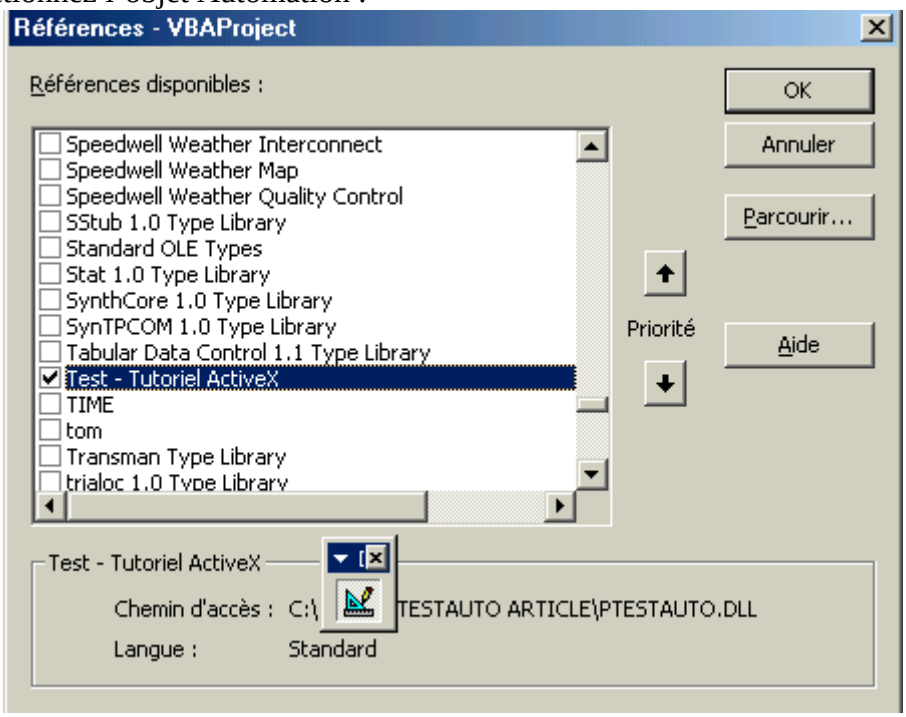
Sous Excel, créez un bouton dans une feuille, mettez dans son événement OnClick un appel à une subroutine CallTest que vous implémentez dans un **module**[2].

Sous VBA, il faut importer l'objet Automation. Cliquez sur le menu **Outils** puis **référence**.

[2] L'utilisation des objets automations ne peut se faire que dans les modules sous VBA.

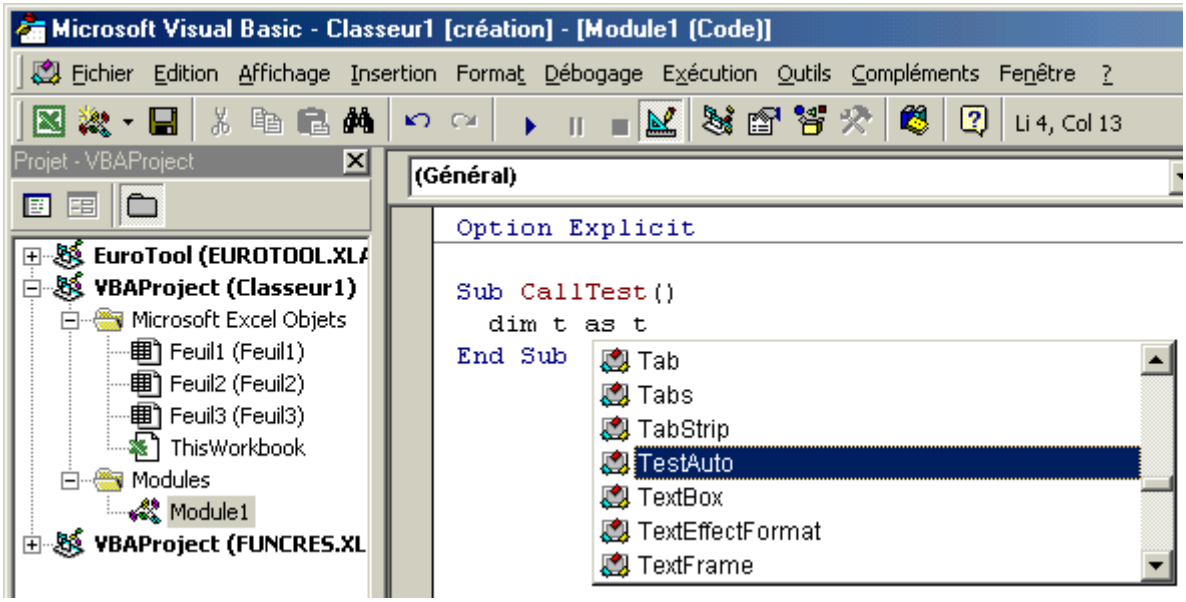


Dans la liste sélectionnez l'objet Automation :



puis cliquez sur OK.

Si vous avez bien fait les choses lorsque vous déclarez les variables, vous devez voir dans la liste le nouvel objet TestAuto :



Enfin, tapez le code exemple ci-dessous :

```
Option Explicit

Sub CallTest()
  Dim t As TestAuto
  Set t = New TestAuto

  Dim i As Long
  Dim v() As Double
  ReDim v(0 To 9)
  For i = 0 To 9
    v(i) = i
  Next i


  t.Donnees = v
  Range("A1") = t.Moyenne


  Set t = Nothing
End Sub
```

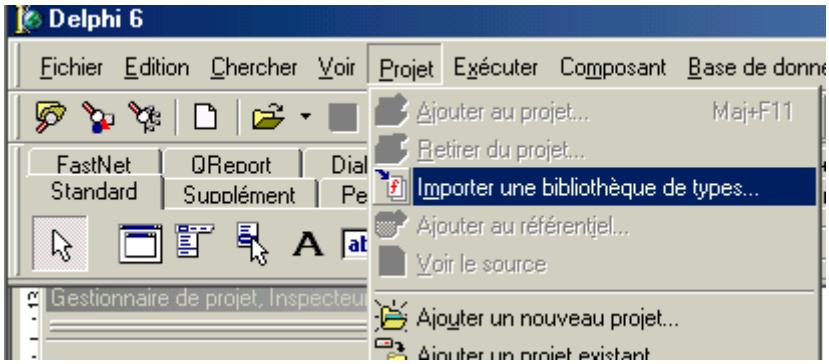
C’est terminé. Cliquez sur le bouton et testez.

III.Interface avec Delphi

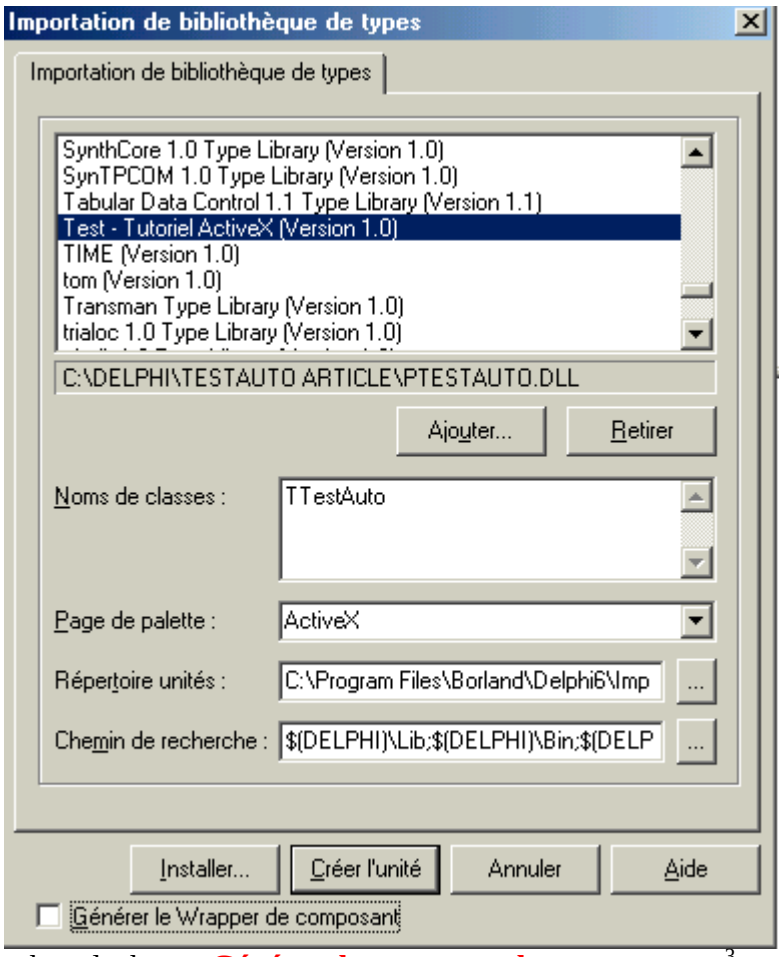
L’interface avec Delphi est très différente de celle de Visual Basic.
Il est nécessaire d’importer la bibliothèque de types et d’éventuellement wrapper les objets sous la forme de composants. Pour ma part, je ne crée pas de composants afin de ne pas alourdir inutilement mes programmes lorsque les objets ne sont pas visuels.

A. Importation de la bibliothèque de types

Fermez toutes vos applications sous Delphi (ce n’est pas vraiment nécessaire mais c’est plus sûr). Puis cliquez sur **Projet** et **importer une bibliothèque de types** :



Sélectionnez l'objet Automation dans la liste



Décochez la case à cocher du bas « **Générer le wrapper de composant** »[3] et cliquez sur le bouton **Installer**. Mettez l'unité dans l'un de vos paquets de composants qui va être recompiler pour l'occasion. Fermez tout et créez une nouvelle application. L'unité a pour nom PTest_TLB et se trouve dans le répertoire Imports de Delphi 6.

B. Création des objets

Apposez un bouton et créez un événement OnClick. Que vous coderez dans l'exemple comme suit :

[3] Il est recommandé de générer le Wrapper de composants au moins une fois pour lire le code automatiquement généré par Delphi.

```
implementation
uses PTest_TLB; // Ne pas oubliez de rajouter dans la clause uses l'unité PTest_TLB
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var i : integer;
  t : ITestAuto; // déclaration de l'interface
  v : OleVariant;
begin
  try
    t := CoTestAuto.Create; // la création de l'objet renvoie une interface !!!
  except
    ShowMessage('L''objet est mal installé');
    exit;
  end;
  v := varArrayCreate([0,9],varDouble); // Création du tableau
  for i:=0 to 9 do v[i] := i;

  t.Donnees := v; // passage de propriété
  Edit1.Text := FloatToStr(t.Moyenne); // calcul et affichage de la moyenne

  t := nil; // Il n'y a pas de free ! on assigne l'adresse nil !
end;
```

N'oubliez pas de rajouter dans vos uses l'unité importée : PTest_TLB.
ATTENTION : Rajouter à la fin de la procedure la commande :

varClear(v);

afin de libérer la mémoire allouer à l'OleVariant v.

IV. Conclusion

A partir de ce tutorial, il devrait vous être possible de programmer n'importe quel objet Automation ou COM. L'apprentissage sera assez long mais les possibilités de vos applications seront nettement étendues. Vous pourrez programmez sous l'architecture n-tiers et vos programme pourront communiquer entre eux sur un réseau sans trop de difficulté. Au sein des objets automation on peut gérer des TForm assez facilement. Il suffira que vous découvriez la technique pour le faire sans bug ce qui m'a pris bien 2 jours de travail à l'époque.