

Création d'un visualiseur de debuggage avec Visual Studio .Net 2005

par [MORAND Louis-Guillaume](#)

Date de publication : 17/05/2005

Dernière mise à jour :

Tutoriel pas à pas expliquant le principe, l'utilisation et le développement d'un visualiseur de debuggage avec Visual Studio .Net 2005

Avant-Propos

- 1 - Le visualiseur de débogage
 - 1.1 - Présentation du visualiseur de debuggage
 - 1.2 - Les visualiseurs de debuggage de base
- 2 - Mon premier visualiseur de debuggage
 - 2.1 - Introduction
 - 2.2 - Développement du visualiseur
 - 2.3 - Débogage du visualiseur
 - 2.4 - Installation du visualiseur
- 4 - Visualiseur avancé
 - 4.1 - Préparation
 - 4.2 - La classe personne
 - 4.3 - Le visualiseur
 - 4.4 - L'application de test
 - 4.5 - Installation et test

Conclusion

Remerciements et liens

Avant-Propos

Le visualiseur de débogage est l'une des nouveautés de la prochaine version de Visual Studio .Net. Cet article a donc été réalisé avec la beta 2 de Visual Studio .Net 2005 et des changements peuvent être apportés dans la version finale. Je tiens également à porter l'attention sur le fait que certains articles de la MSDN inclus dans la bêta 2 sont erronés mais que la version en ligne est à jour: préférez donc son utilisation.

Cet article s'adresse à des personnes ayant un minimum de connaissances du langage .Net (serialisation, attributs, etc). De plus, cet article comportera trois parties, une de présentation, une partie sur un exemple simple, et une partie abordant les classes personnalisées.

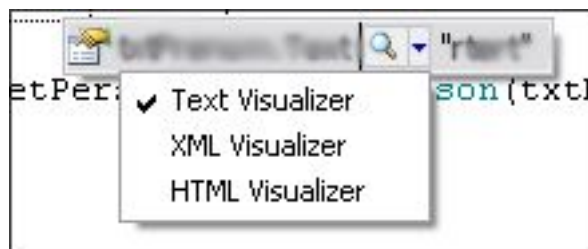
1 - Le visualiseur de débogage

Dans le cycle du développement d'une application, le débogage prend une part importante. En effet, c'est durant ce cycle que le développeur doit analyser le comportement de son programme et l'état changeant des variables et des méthodes qu'il contient. Jusqu'à aujourd'hui, le mode debug se faisait à l'aide de breakpoints et d'espions qui montraient de façon brute la valeur des variables.

1.1 - Présentation du visualiseur de débogage

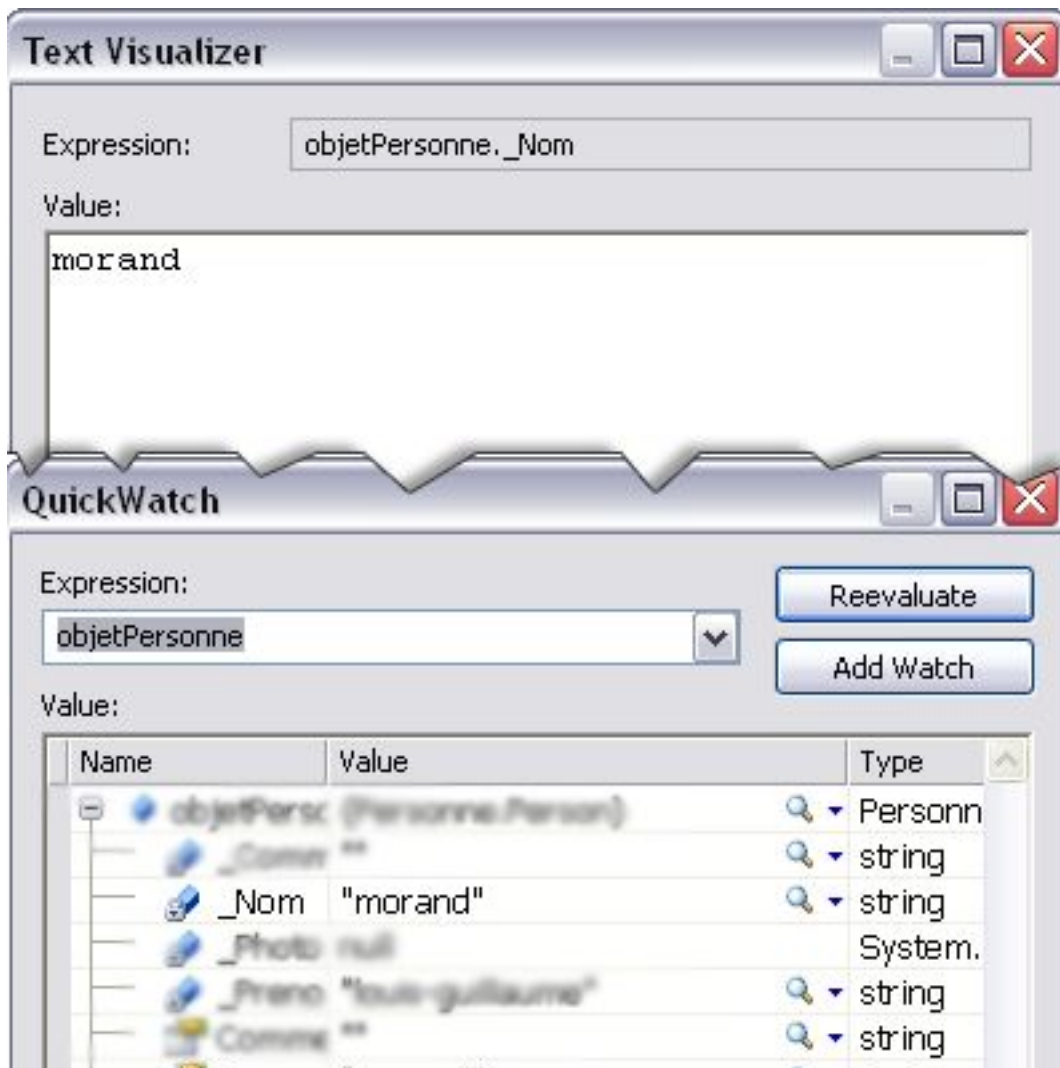
1.2 - Les visualiseurs de débogage de base

Visual Studio .Net 2005 (beta 2?) propose trois visualiseurs de débogage: le visualiseur de texte, celui de XML et celui de HTML.

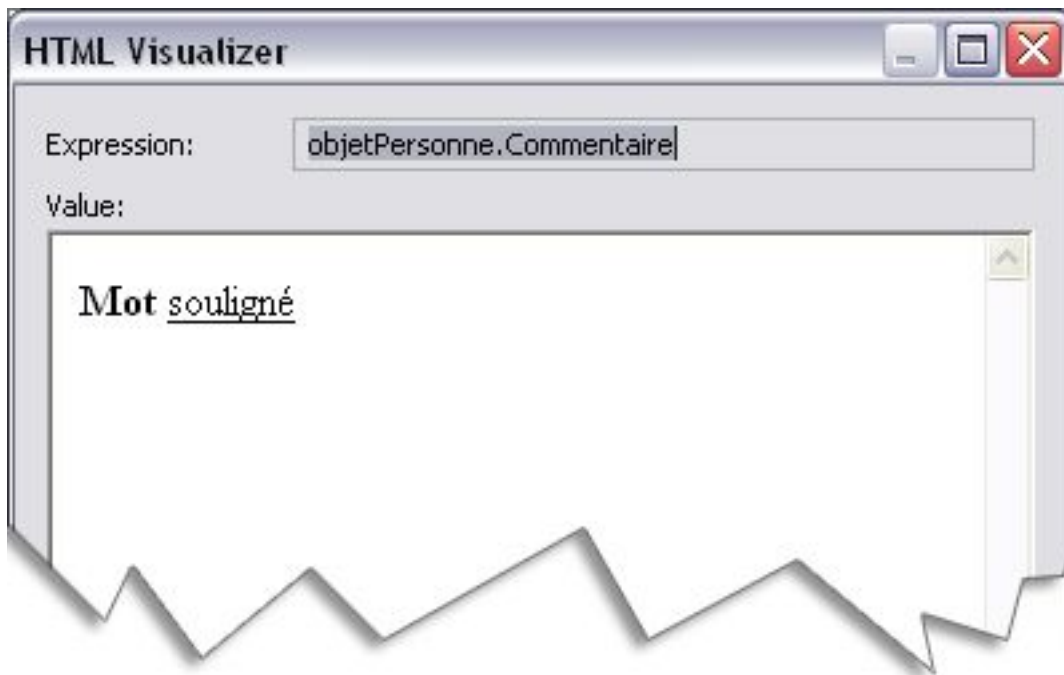


Les différents visualiseurs de base

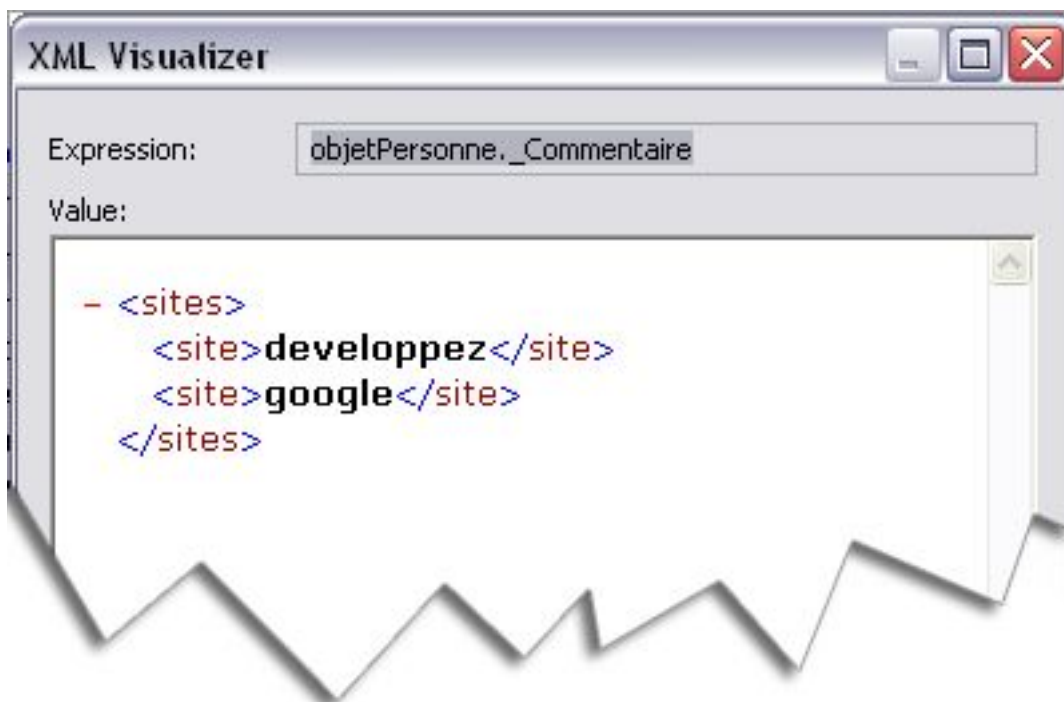
Nous avons tout d'abord, le visualiseur de texte, ce dernier permet entre autre de voir la valeur d'une chaîne de caractères. Ce "watcher" correspond au watcher que l'on avait dans les versions précédentes. Son affichage est limité et est identique à celui intégré dans Visual Studio: le quickwatch.



Parmi les nouveautés, nous avons donc le visualiseur HTML. Celui permet de visualiser une chaîne de caractères contenant du code HTML.



Et enfin, nous avons le visualiseur de texte XML. Comme Internet Explorer, il ouvre et indente le code XML automatiquement pour en faire un affichage clair, coloré et compréhensible pour l'utilisateur:



2 - Mon premier visualiseur de debuggage

2.1 - Introduction

Nous allons maintenant créer notre premier visualiseur. Celui-ci sera un visualiseur très simplifié qui permettra de voir la valeur d'un objet de type string. Nous prenons ici un exemple extrêmement simple pour découvrir peu à peu les difficultés du développement d'un visualiseur. Le chapitre suivant sera plus avancé.

2.2 - Développement du visualiseur

Ouvrez Visual Studio .Net 2005 et créez un Nouveau Projet > C# > Class library (bibliothèque de classe). Nommez votre projet : MonPremierVisualiseur.

Dans l'explorateur de solution, faites bouton droit sur le fichier Class1.cs, puis renommez le en Visualiseur. Une des nouveautés de Visual Studio .Net 2005 sera de renommer également le nom de la classe interne à ce fichier. Votre fichier doit alors contenir le texte suivant:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace MonPremierVisualiseur
{
    public class Visualiseur
    {
    }
}
```

Ajoutez maintenant une référence à votre projet, en cliquant sur Add reference > Onglet .Net > Microsoft.VisualStudio.DebuggerVisualizers.

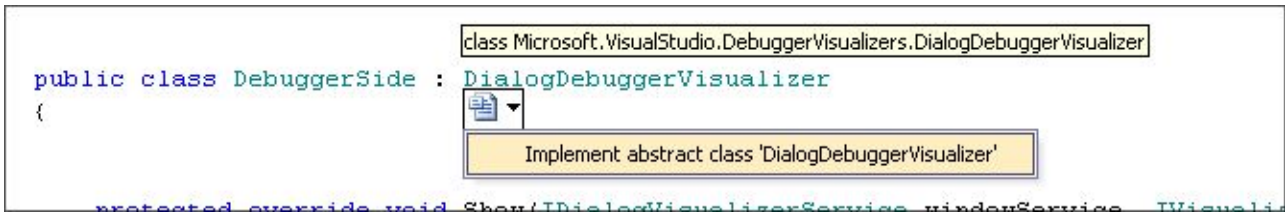
Ajoutez ensuite la clause *using* correspondante:

```
using Microsoft.VisualStudio.DebuggerVisualizers;
```

Nous voulons maintenant que notre classe hérite de la classe de base *DialogDebuggerVisualizer*

```
public class Visualiseur : DialogDebuggerVisualizer
{
}
```

Il nous faut implémenter les méthodes nécessaires au fonctionnement du visualiseur. Visual studio 2005 nous aide alors, à l'aide d'un SmartTag. Cliquez sur le mot *DialogDebuggerVisualizer* et laissez le curseur dessus. Un SmartTag devrait alors vous proposer d'implémenter la méthode manquante:



Ceci doit normalement implémenter la méthode suivante, point d'entrée de votre visualiseur.

```

protected override void Show(IDialogVisualizerService windowService, IVisualizerObjectProvider
objectProvider)
{
}

```

Cette méthode prend en argument deux objets. Ces objets seront transmis automatiquement par Visual Studio, vous n'aurez à vous soucier que de la récupération des données qu'ils contiennent.

Nous avons donc:

l'objet IDialogVisualizerService qui est utilisé pour fournir des informations spécifiques au type d'interface utilisateur que ce visualiseur doit faire apparaître.

l'objet IVisualizerObjectProvider, qui est l'interface à travers laquelle sont envoyés et reçus des streams entre le visualiseur (debugger-side) et le programme débogué (program-side). Cette interface contient une méthode GetObject() qui permet de récupérer l'objet "sélectionné" dans le programme débogué.

Nous voulons que notre visualiseur se "déclenche" pour les objets de type *string*. Ajoutons donc l'attribut suivant qui décrit notre visualiseur

```

[assembly: System.Diagnostics.DebuggerVisualizer(
typeof(MonPremierVisualiseur.Visualiseur),
Target = typeof(System.String),
Description = "Mon visualiseur")]

```

Le type est celui de notre classe (rien de plus logique), la cible correspond à l'objet pour lequel le visualiseur se "déclenchera" ou plutôt sera accessible. Et enfin, la description est le texte par lequel sera décrit notre visualiseur.

Ajoutez donc le code suivant, pour récupérer la valeur de la chaîne "surveillée".

```

MessageBox.Show(objectProvider.GetObject().ToString());

```

La méthode GetObjet() permet donc de récupérer l'objet sérialisé pour passer dans le stream du debugger. Ici donc, nous récupérons l'objet et nous le "castons" en *string*; chose peu gênante puisque notre visualiseur se limitait aux objets de type *string*. Ce code est très limité et ne gère pas les null exceptions par exemple. De plus, son fonctionnement reste simple mais nous allons voir comment l'étudier (déboguer) de plus près.

2.3 - Débogage du visualiseur

Un visualiseur de débogage est utile pour déboguer un projet mais lui même doit être débogué pendant son

développement. Trois solutions s'offrent à nous: compiler, installer et vérifier que tout marche, ou alors debugguer une seconde instance de Visual Studio (solution lourde) ou encore utiliser un composant créé exprès pour le debuggage des visualiseurs. Nous allons pour cela créer une application de test et appeler notre visualiseur SANS être en mode debug.

Dans l'explorateur de solution, cliquez sur votre solution et ajoutez un nouveau projet > Application console C#

Définissez ce projet comme projet de démarrage (lors de la compilation).

Dans la méthode main(), ajoutez un objet de type *string* et instanciez le:

```
string maChaine = "www.developpez.com";
```

Ajoutez la référence à la librairie *Microsoft.VisualStudio.DebuggerVisualizers*.

Collez alors le texte suivant:

```
VisualizerDevelopmentHost host = new VisualizerDevelopmentHost(  
maChaine, typeof(MonPremierVisualiseur.Visualiseur));  
host.ShowVisualizer();
```

De cette façon, nous appelons directement le visualiseur comme si nous étions en train de debugguer une autre instance de Visual Studio. Ici, en appelant la méthode *ShowVisualizer()*, nous appelons la méthode *Show()* de la classe de notre visualiseur. Nous pouvons alors debugguer ce fichier classe comme n'importe quelle autre application.

2.4 - Installation du visualiseur

Le visualiseur de debuggage est une extensibilité de Visual Studio 2005 et le but est qu'il soit accessible quelque soit le projet ouvert. Nous allons donc l'installer "dans" Visual Studio .Net 2005.

Générer donc votre projet (ctrl+maj+B), puis allez dans le repertoire `\MonPremierVisualiseur\bin\debug`, prenez le fichier *MonPremierVisualiseur.dll* et copiez dans l'un des repertoires destinés à cet effet:

- Mes documents\Visual Studio\Visualizers

ou

- Installation de VS 2005\Common7\Packages\Debugger\Visualizers

Je vous conseille personnellement le premier choix afin d'y accéder plus rapidement et de pouvoir plus simplement déplacer vos fichiers sur une autre machine.

Votre visualiseur est installé, vous pouvez le tester depuis n'importe quelle application. Ouvrez donc un projet, placez des breakpoints et lancez le en mode debug. Trouvez ensuite un objet de type *string* et placez votre curseur

dessus. Le SmartTag devrait alors apparaître et en cliquant sur la loupe, vous devriez voir votre visualiseur parmi les visualiseurs de base.

4 - Visualiseur avancé

Nous avons vu comment créer un visualiseur de débogage pour un objet de type string, et qui aurait pu être un entier ou une image. Néanmoins, même si les articles de ce type pullulent sur la toile, ils ne sont pas d'après moi d'une utilité très convaincante. En effet, nous savons tous déjà, lancer une application en mode debug pour avoir la valeur d'une chaîne de caractères (Quickwatch), ou dans l'exemple de l'image, nous n'avons pas régulièrement besoin, de connaître l'état d'un objet image en cours d'exécution.

Je pense donc, qu'il est plus courant d'avoir à debugger des objets ou des classes personnelles, plus encore, des classes contenant un grand nombre d'objets de types différents. Ainsi, pour cet exemple "avancé", nous prendrons comme exemple, une classe personnelle: la classe Person. Cette classe représente une entité, ayant un nom(*string*), un prénom(*string*), un commentaire(*string*) et une photo(*Image*).

4.1 - Préparation

Voici le code de la classe Person:

Classe Person

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;

namespace Personne
{
    [Serializable]
    public class Person
    {
        #region Variables
        private string _Nom;
        private string _Prenom;
        private Image _Photo;
        private string _Commentaire;
        #endregion

        #region Accesseurs
        public Image Photo
        {
            get { return _Photo; }
            set { _Photo = value; }
        }
        public string Prenom
        {
            get { return _Prenom; }
            set { _Prenom = value; }
        }
        public string Nom
        {
            get { return _Nom; }
            set { _Nom = value; }
        }
        public string Commentaire
        {
            get { return _Commentaire; }
            set { _Commentaire = value; }
        }
        #endregion

        #region Constructeurs
        public Person()
        {
        }
        public Person(string nom, string prenom, string commentaire, Image image)
        {
            this._Nom = nom;
            this._Prenom = prenom;
            this._Commentaire = commentaire;
        }
    }
}
```

```
Classe Person
    this._Photo = image;
}
#endregion
}
```

Attaquons nous au développement. Créez un nouveau projet de type bibliothèque de classes, que vous nommerez PersonneVisualiseur. Dans votre solution, créez un deuxième projet de bibliothèque de classes que vous nommerez Personne, dans lequel, vous remplacerez le fichier autogénéré par le fichier person.cs cité ci-dessus. Créez un troisième projet windows form que vous nommerez comme vous souhaitez. Vous avez donc:

- projet du visualiseur : PersonneVisualiseur (bibliothèque de classe)
- projet de la classe Person : Personne (bibliothèque de classe)
- projet de test : Nom que vous voulez (Application Windows)

Passons aux références:

dans le projet PersonneVisualiseur, ajoutez une référence :

- à Microsoft.VisualStudio.DebuggerVisualizers
- à System.Windows.Forms (sera utilisé pour l'affichage)
- au projet Personne

dans le projet de test (application windows)

- à Microsoft.VisualStudio.DebuggerVisualizers
- au projet Personne

Les projets sont prêts, vous vous rendrez vite compte du pourquoi de cette séparation.

4.2 - La classe personne

Pourquoi la définir dans un projet tiers? Tout simplement parce que le projet que vous souhaitez déboguer doit y faire référence. Nous aurions donc pu intégrer cette classe dans le projet, mais nous avons également besoin d'y faire référence dans notre visualiseur. Cette méthode est alors plus simple. Qui plus est, les visualiseurs de débogage sont généralement utilisés pour des classes personnelles que vous réutiliser de projet en projet. C'est alors la meilleure solution.

Ensuite, c'est une classe toute simple, dans laquelle j'ai défini les accesseurs pour chacune des propriétés même si ces propriétés n'étaient pas forcément nécessaires pour le visualiseur.

La dernière chose à noter, est l'attribut [serializable] placé avant la classe. Cet attribut est nécessaire, en effet, comme vu dans le chapitre précédent, l'objet est serialisé puis passé dans un stream avant d'être accessible par le visualiseur. Notre classe doit donc pouvoir être sérialisable.

4.3 - Le visualiseur

Le visualiseur reste assez simple. A la place du messageBox affichant la chaîne de caractères (vu dans la partie précédente), nous ouvrirons ici une form (d'où la référence nécessaire), dans laquelle, nous afficherons les propriétés de l'objet Person.

Nous créons donc une form simple, dans laquelle nous ajoutons un constructeur prenant en paramètre un objet de type Person: objet que nous récupérerons via le visualiseur avant de le repasser à cette fenêtre.

```
public FormDebug(Person objPers)
{
    InitializeComponent();
    txtNom.Text = objPers.Nom;
    txtPrenom.Text = objPers.Prenom;
    txtCommentaire.Text = objPers.Commentaire;
    pictureBox1.Image = objPers.Photo;
}
```

Dans le fichier principal de notre visualiseur, nous modifions maintenant la méthode Show() afin de "transmettre" l'objet récupéré à notre fenêtre:

```
protected override void Show(IDialogVisualizerService windowService, IVisualizerObjectProvider
objectProvider)
{
    FormDebug frm = new FormDebug((Person)objectProvider.GetObject());
    frm.ShowDialog();
}
```

Maintenant, chose **très importante**, nous allons modifier l'attribut de notre visualiseur pour qu'il se déclenche pour les objets de type Person:

```
[assembly: System.Diagnostics.DebuggerVisualizer(
typeof(FirstVisualizer.DebuggerSide),
Target = typeof(Personne.Person),
Description = "Mon visualiseur")]
```

4.4 - L'application de test

Nous ferons ici une application tout ce qu'il y a de plus simple, et nous aurions pu faire une application console, mais j'ai préféré faire une interface graphique afin de remplir nous même l'objet Person, car nous pourrions avoir à le faire dans une vraie application utilisant cet objet. Notre application ressemble à cela:



Nous renseignons donc les champs, choisissons une image et cliquons sur le bouton Créer pour créer un objet Person dont les propriétés sont instanciées. Il ne nous reste plus qu'à lancer cette application en mode debug pour pouvoir tester notre visualiseur.

4.5 - Installation et test

Générez la solution, et copiez la dll du visualiseur dans le dossier des visualiseurs (cf chapitre 2).

Placez un point d'arrêt à la ligne suivante de la création de notre objet Person.

Lancez l'application en mode debug, remplissez les champs (Nom, prénom, photo, etc), cliquez sur OK et Visual Studio va alors se positionner sur le point d'arrêt:



Cliquez sur "Mon visualiseur", et une fenêtre s'ouvre alors:



Nous venons donc, pour une classe totalement personnelle, créer un visualiseur et nous pouvons en cours d'exécution, tester la valeur de chaque propriété de notre objet Person, notamment de la propriété *photo* de type Image, qu'il nous serait en temps normal, impossible de visualiser.

Conclusion

Vous savez maintenant comment créer votre propre visualiseur. Sachez que le développement d'un visualiseur est très rapide, néanmoins à moins d'en créer un pour des types de base du framework, ils sont limités à un et un seul type d'objet. L'exemple avancé que nous venons de voir est alors utile dans le cas d'une classe que vous réutilisez plus ou moins régulièrement au long de vos projets ou alors une classe difficilement debuggable.

Remerciements et liens

J'aimerais remercier tout particulièrement [Freegreg](#) pour ses corrections apportées à cet article.

Voici les sources des deux visualiseurs vus tout du long de cet article.

[Sources visualiseur de string](#) (6ko)

[Sources visualiseur personnel](#) (56ko)

Je vous encourage également à lire l'article suivant vous montrant un autre exemple simple mais utile :

[Un visualiseur pour debugguer des objets de type DataSet](#) par Julia Lerman