

[Forums](#) [Tutoriels](#) [Magazine](#) [FAQs](#) [Blogs](#) [Chat](#) [Newsletter](#) [Études](#) [Emploi](#)[Club](#) [Contacts](#)p  
u  
b  
l  
i  
c  
i  
t  
é[Accueil](#) [ALM](#) [Java](#) [.NET](#) [Dév. Web](#) [EDI](#) [Programmation](#) [SGBD](#) [Office](#) [Solutions d'entreprise](#) [Applications](#) [Mobiles](#)  
[Systèmes](#) [.NET](#) [Visual Studio](#) [ASP.NET](#) [C#](#) [Visual Basic.NET](#) [Windows Phone](#) [Microsoft Azure](#)[FORUMS .NET](#)[FAQs .NET](#)[TUTORIELS .NET](#)[VIDÉOS .NET](#)[SOURCES .NET](#)[LIVRES .NET](#)[OUTILS .NET](#)[AZURE DEV CAMP](#)

## Les tests unitaires avec Nunit.


### Table des matières

- Remerciements
- I. Introduction
- II. Installation
  - i. Vous utilisez Mono
  - ii. Vous utilisez SharpDevelop
  - iii. Dans les autres cas
- III. Présentation
  - A. Le mode ligne de commande
  - B. L'interface graphique
    - I. Ouvrir un assembly contenant les tests
    - II. Ouvrir un projet Visual Studio contenant les tests
    - III. L'éditeur de projet
    - IV. L'interface proprement dite
- IV. Intégration dans visual studio
- V. La procédure de test
  - a. Généralités
  - b. Avertissement
  - c. La classe à tester
  - d. Le programme de test
    - 1. Définition de la classe réalisant les tests unitaires
    - 2. Le test ne doit pas générer d'exception
    - 3. Vérifier qu'une valeur est bien la valeur attendue
    - 4. Le résultat de l'expression doit être vrai
    - 5. Le résultat de l'expression doit être faux
    - 6. Vérifier qu'une variable ou une propriété n'est pas assignée
    - 7. Vérifier qu'une variable ou une propriété a été assignée
    - 8. Vérifier que 2 variables ou propriétés pointent sur la même référence.
    - 9. Vérifier qu'une exception est bien générée
    - 10. Le test explicite
    - 11. Ignorer des tests
    - 12. Regrouper les tests dans des catégories
- VI. Conclusion

L'utilité des tests unitaires n'est plus à démontrer et Nunit est l'un des meilleurs outils en la matière pour la plateforme .NET. Dans cet article, nous verrons comment utilise Nunit pour faire nos tests unitaires. (Version Pdf)

Article lu 6397 fois.

## L'auteur

Jean-Alain Baeyens 

## L'article

Publié le 28 avril 2005

Version PDF Version hors-ligne

ePub, Azw et Mobi

## Liens sociaux



Partager

## Remerciements▲

Je remercie mon épouse Dominique ainsi que Louis-Guillaume Morand pour la relecture de ce tutorial. Je remercie également les contributeurs de Nunit pour leur excellent travail. Vous trouverez le site officiel de Nunit ici. Je remercie également tous ceux qui ont contribué à la très bonne documentation du produit qui malheureusement n'existe actuellement pas en français.

## I. Introduction▲

L'utilité des tests unitaires n'est plus à démontrer. Ils permettent d'obtenir un code plus robuste et contrôlent la non-régression lors de mises à jour de vos classes. Une fois les tests construits, vous pouvez les réexécuter sans efforts. Nunit est probablement l'outil le plus connu pour la plateforme dotnet, et est un produit gratuit sous licence open source. Vous pouvez trouver le détail de la licence ici. La version de Nunit présentée ici est la version 2.2.0.0

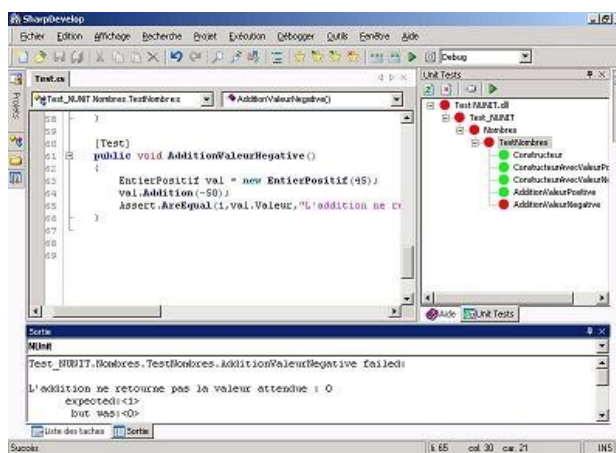
## II. Installation▲

### i. Vous utilisez Mono▲

Mono est livré avec une version pré-installée de Nunit. Vous n'avez donc rien à faire.

### ii. Vous utilisez SharpDevelop▲

SharpDevelop intègre complètement Nunit dans son IDE. Il y reçoit une interface graphique spécifique pour être complètement intégré à l'interface de SharpDevelop.



Nunit dans l'interface de SharpDevelop

### iii. Dans les autres cas▲

Vous pouvez télécharger la dernière version de Nunit ici. Ensuite il suffit de lancer le programme d'installation (NUnit-2.2.0.msi). Vous recevrez alors la succession d'écrans suivante:



### Présentation



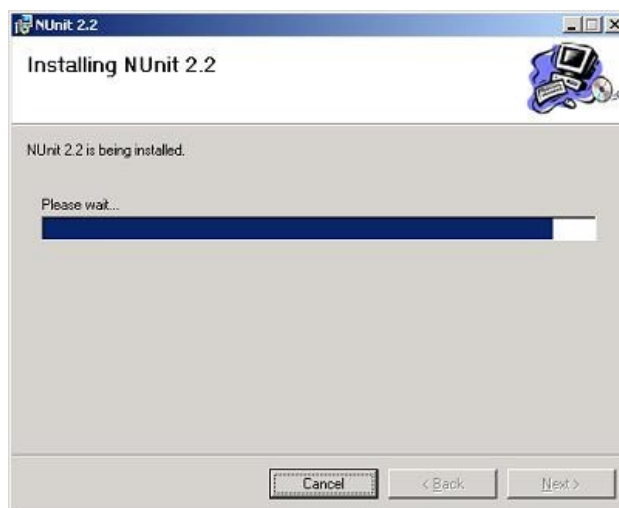
### Chemin d'installation



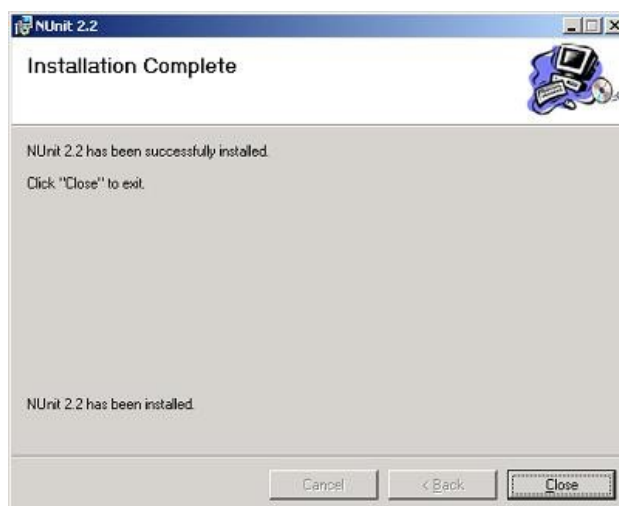
### Accord de license



Confirmation



Progression. La barre défile 2 fois.



Installation terminée

### III. Présentation ▲

#### A. Le mode ligne de commande ▲

Vous pouvez toujours opter pour l'exécution de NUnit en mode console. Cette option peut être intéressante pour exécuter NUnit en mode batch. Le résultat est alors produit dans un fichier XML que vous pouvez exploiter ultérieurement.

Pour démarrer les tests en mode console, vous devez utiliser la commande "NUNIT-CONSOLE" qui vous trouverez dans le répertoire bin de NUnit. Vous devez faire suivre cette commande par le ou les assemblies qui contiennent les tests. Vous pouvez également spécifier le nom d'un projet NUnit en lieu et place du, des assemblies.

Exemple  
Sélectionnez

```
NUNIT-CONSOLE MonAssembly.dll /xml=TestResult.Txt
```

La commande dispose de 14 options permettant de l'adapter au mieux à votre usage. Vous pouvez ainsi spécifier le nom des fichiers de sortie, la liste des catégories,...

Pour obtenir la liste complète des options, il suffit de taper la commande sans paramètre.

## B. L'interface graphique ▲

L'interface graphique est le mode le plus couramment utilisé. Il vous permet de voir immédiatement les résultats des tests.

### I. Ouvrir un assembly contenant les tests ▲

Rien de plus simple, vous choisissez l'option "Open" dans le menu, vous choisissez l'assembly qui contient les tests et ceux-ci apparaissent dans la fenêtre de test. Il ne vous reste qu'à les exécuter.

Si vous souhaitez exécuter simultanément des tests contenus dans différents assemblies, il vous suffit d'ajouter les autres assemblies en utilisant l'option "Add Assembly". Vous pouvez alors enregistrer le projet NUnit et ainsi ouvrir le projet la fois suivante pour revenir à la même configuration.

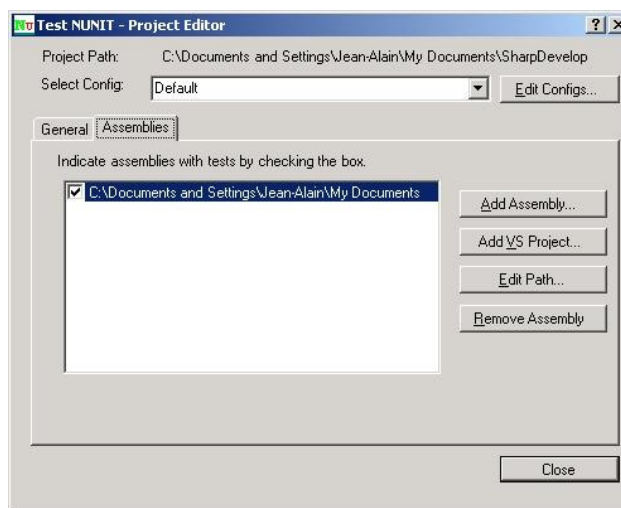
Par défaut, NUnit ouvre automatiquement le dernier projet ou assembly utilisé. Vous pouvez modifier ce comportement via la fenêtre des options.

### II. Ouvrir un projet Visual Studio contenant les tests ▲

Si vous utilisez Visual Studio 2003, vous pouvez ouvrir la solution ou le projet qui contient les tests via l'option "Add VS project". Cette option n'est disponible que si vous avez déjà ouvert un assembly, un projet NUnit ou créé un nouveau projet NUnit.

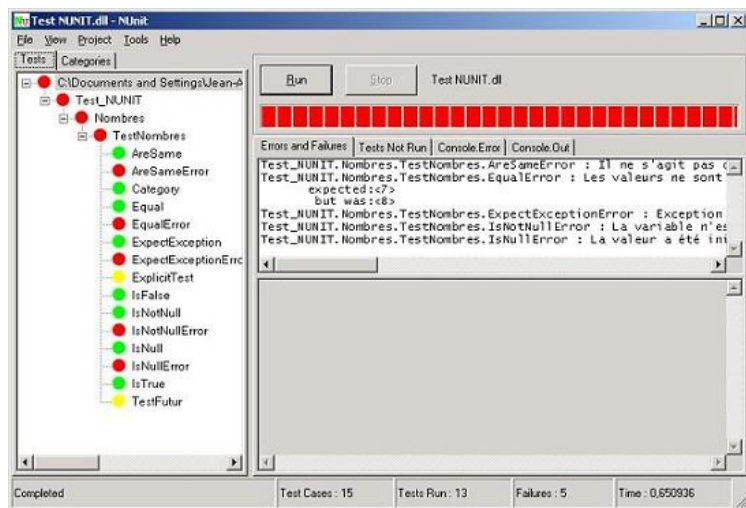
### III. L'éditeur de projet ▲

L'éditeur de projet vous permet de gérer votre projet NUnit. C'est par ce moyen que vous pouvez enlever un assembly.



L'editeur de projet

### IV. L'interface proprement dite ▲



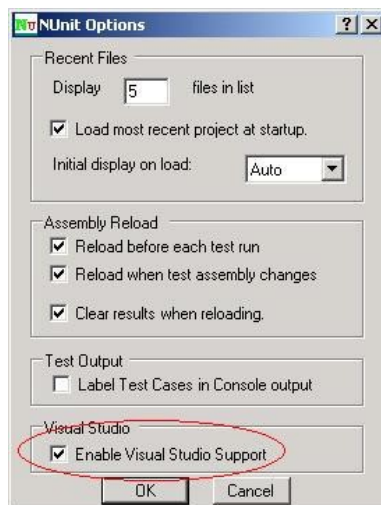
L'interface graphique de NUnit

La fenêtre de gauche vous donne la liste des tests. Après exécution, les points sont colorés. Le point vert indique que le test s'est bien déroulé, le point rouge indique une erreur et le point jaune un test qui n'a pas été exécuté. La fenêtre de droite vous donne des informations concernant l'exécution des tests.

#### IV. Intégration dans visual studio ▲

Nunit peut être intégré à Visual Studio mais il s'agit d'une intégration limitée. Pour réaliser cette intégration, vous devez ajouter Nunit dans les "custom tools". Vous spécifiez Nunit.Gui comme command. Vous utilisez \$(TargetPath) comme argument et \$(TargetDir) comme répertoire initial. La documentation de Nunit renseigne ces valeurs pour un projet C# mais à priori ils devraient convenir également pour VB.NET.

Dans l'interface proprement dit de Nunit, cochez l'option "Visual Studio support".



Fenêtre des options

#### V. La procédure de test ▲

##### a. Généralités ▲

Bien que cela ne soit pas le but de l'article, il est bon de rappeler quelques règles de base concernant les tests unitaires. Les tests doivent être le plus complets possible. Il est toutefois inutile de tester l'évidence. Le code permettant de réaliser les tests doit être le plus simple possible. Idéalement, vos tests doivent être préparés avant de réaliser la classe elle-même. Une bonne idée est de préparer le squelette de la classe (définition des propriétés et des méthodes) ensuite de coder vos tests unitaires et enfin de coder les méthodes de votre classe.

##### b. Avertissement ▲

Le code présenté aussi bien pour la classe que pour la procédure de test est volontairement simplissime, erroné et ne respecte en rien la bonne pratique du test unitaire. Il est simplement là pour vous présenter différentes possibilités offertes par NUnit.

Il ne s'agit pas d'une description exhaustive de toutes les possibilités mais les exemples proposés ici doivent couvrir la majorité de vos besoins.

Le code exemple des tests présentés ci-après ne requiert généralement pas d'explication complémentaire et vous est présenté tel quel.

## c. La classe à tester▲

Sélectionnez

```
using System;

namespace Nombres
{
    public class EntierPositif
    {
        private int valeur;
        /// <summary>
        /// Valeur de notre EntierPositif. On génère une exception
        /// si cette propriété reçoit un nombre négatif.
        /// </summary>
        public int Valeur {
            get {
                return valeur;
            }
            set {
                if (value >= 0){
                    valeur = value;
                }
                else{
                    throw new Exception("L'assignation d'une val
                }
            }
        }

        /// <summary>
        /// Constructeur par défaut.
        /// </summary>
        public EntierPositif(){
            valeur = 0;
        }

        /// <summary>
        /// Constructeur par défaut. La valeur assignée est 0.
        /// </summary>
        /// <param name="valinit">
        /// Indique si la valeur doit être initialisée à 0 ou non.
        /// </param>
        public EntierPositif(bool valinit){
            if (! valinit)
            {
                valeur = 0;
            }
        }

        /// <summary>
        /// Le constructeur reçoit la valeur initiale en paramètre.
        /// Si la valeur est négative, le constructeur génère une exception.
        /// </summary>
        /// <param name="val"></param>
        public EntierPositif(int val){
            Valeur = val;
        }

        public void Soustraction(int val){
            // A développer
        }
    }
}
```

## d. Le programme de test▲

### 1. Définition de la classe réalisant les tests unitaires▲

Sélectionnez

```
using System;
using NUnit.Framework;
using Nombres;
namespace Test_NUNIT.Nombres
{
    [TestFixture]
    public class TestNombres
    {
        // contient une série de méthodes définies comme ceci:
        // [TEST]
        // public void MonTest()
        // {
        //     ...
        // }
    }
}
```

L'attribut "TestFixture" définit que la classe contient des tests. Vous pouvez parfaitement avoir plusieurs classes marquées par "TestFixture".

## 2. Le test ne doit pas générer d'exception ▲

Sélectionnez

```
[Test]
public void SimpleExecution()
{
    EntierPositif x = new EntierPositif(7);
    x.Valeur --;
    x.Valeur = x.Valeur * 2;
    x.Valeur ++;
}
```

## 3. Vérifier qu'une valeur est bien la valeur attendue ▲

Sélectionnez

```
[Test]
public void Equal()
{
    EntierPositif x = new EntierPositif(7);
    EntierPositif y = new EntierPositif(4+3);
    Assert.AreEqual(x.Valeur,y.Valeur,"Les valeurs ne sont pas égales");
}
```

## 4. Le résultat de l'expression doit être vrai ▲

Sélectionnez

```
[Test]
public void IsTrue()
{
    EntierPositif x = new EntierPositif(7);
    Assert.IsTrue((x.Valeur > 5), "La valeur n'est pas correct");
}
```

## 5. Le résultat de l'expression doit être faux ▲

Sélectionnez

```
[Test]
public void IsTrue()
{
    EntierPositif x = new EntierPositif(7);
    EntierPositif y = x.Valeur - 1;
    Assert.IsFalse((x.Valeur == y.Valeur), "La valeur n'est pas correct");
}
```

## 6. Vérifier qu'une variable ou une propriété n'est pas assignée ▲

Sélectionnez

```
[Test]
public void IsNull()
{
    EntierPositif val = new EntierPositif(false);
    val = null;
    Assert.IsNull(val,"La valeur a été initialisée");
}
```



```
}
```

## 7. Vérifier qu'une variable ou une propriété a été assignée▲

Sélectionnez

```
[Test]
public void IsNotNull()
{
    EntierPositif val = new EntierPositif();
    Assert.IsNotNull(val, "La variable n'est pas initialisée");
}
```

## 8. Vérifier que 2 variables ou propriétés pointent sur la même référence.▲

Sélectionnez

```
[Test]
public void AreSame()
{
    EntierPositif x = new EntierPositif(7);
    EntierPositif y = x;
    Assert.AreSame(x,y, "Il ne s'agit pas de la même référence");
}
```

## 9. Vérifier qu'une exception est bien générée▲

Sélectionnez

```
[Test]
[ExpectedException(typeof(Exception))]
public void ExpectException()
{
    EntierPositif val = new EntierPositif();
    val.Valeur = -7;
}
```

## 10. Le test explicite▲

La mention "explicit" permet de définir que le test ne sera pas exécuté lors du lancement de la procédure de test. Le test sera exécuté uniquement à la demande, par exemple en double cliquant sur la ligne correspondante dans l'interface graphique. Cette fonctionnalité peut être intéressante pour des tests longs, généralement des boucles, par exemple pour tester la libération des ressources.

Sélectionnez

```
[Test, Explicit]
public void ExplicitTest()
{
    EntierPositif val;
    for (int i=1;i<10000;i++)
    {
        val = new EntierPositif(i);
    }
}
```

## 11. Ignorer des tests▲

Outre la sélection manuelle des tests, vous pouvez choisir d'ignorer certains tests. Les tests ignorés ne seront pas exécutés. Vous pouvez par exemple utiliser cette technique pour ne pas exécuter des tests relatifs à des parties de code pas encore développées. Attention toutefois, le code présent dans le test sera compilé. Il faut donc un minimum de réalisé au niveau de la classe.

On a juste créé le test pour ne pas l'oublier

Sélectionnez

```
[Test]
[Ignore("Pas encore développé")]
public void TestFutur()
{
    // description du test à réaliser
}
```

Le test et le prototype de la classe sont déjà écrit

**Sélectionnez**

```
[Test]
[Ignore("Pas encore développé")]
public void TestFutur()
{
    EntierPositif val = new EntierPositif();
    val.Soustraction(-7);
    Assert.IsTrue(val.Valeur == 0, "L'erreur n'a pas été détectée");
}
```

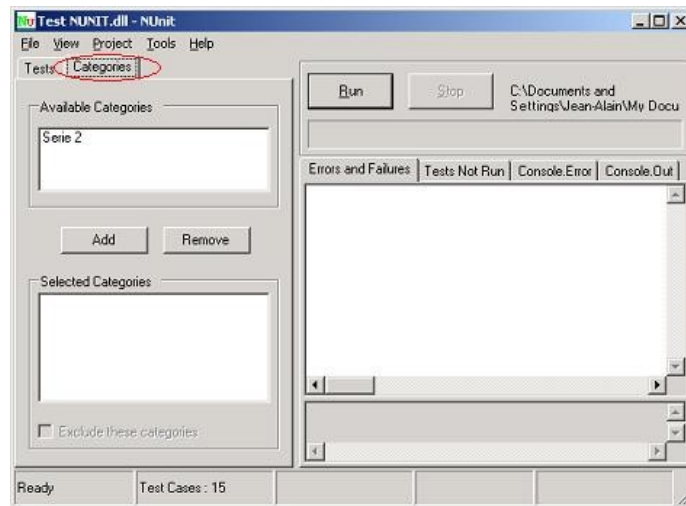
**12. Regrouper les tests dans des catégories ▲**

Il est possible de regrouper les tests dans des catégories.

**Sélectionnez**

```
[Test]
[Category("Serie 2")]
public void Category()
{
    EntierPositif val = new EntierPositif();
    Assert.IsNotNull(val, "La variable n'est pas initialisée");
}
```

Pour pouvoir uniquement exécuter certaines catégories de test, il suffit d'utiliser l'onglet "Catégories" de l'interface.



Utilisation des catégories dans l'interface graphique

**VI. Conclusion ▲**

Grâce à Nunit vous pourrez rendre votre code plus robuste et diminuer le nombre de bugs. Cela vous demande un effort supplémentaire en début de développement mais cet effort sera par la suite vite rentabilisé.

Les sources présentées sur cette page sont libres de droits et vous pouvez les utiliser à votre convenance. Par contre, la page de présentation constitue une œuvre intellectuelle protégée par les droits d'auteur. Copyright © 2005 Jean-Alain Baeyens. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

**Responsable bénévole de la rubrique Microsoft DotNET : Francis Walter - Contacter par email**

Nous contacter    Participez    Hébergement    Informations légales    Partenaire : Hébergement Web

**Copyright © 2000-2015 - www.developpez.com**