Laboratoire Microsoft

Visual Studio 2008 - Tests Unitaires

Lemettre Arnaud

Version 1.0

24 Pages

21/04/2008

EPITA-MTI2009-MS-TUTO-VisualStudio-Test-Unitaire

Propriétés du document

Auteur	**Lemettre Arnaud**
Version	1.0
Nombre de pages	24
Références	EPITA-MTI2009-MS-TUTO-VisualStudio-Test-Unitaire

Historique du document

Date de vision	**Version**	**Auteur**	**Changements**
21/04/2008	0.1	Lemettre	création

Site de référence

description	**url**
Site MTI	http://mti.epita.net/
Blog MTI	http://www.mti.epita.fr/blogs/

Sommaire

Introduction

Dans ce tutorial nous allons aborder le sujet des tests unitaires, et comment les utiliser dans Visual studio 2008.

Beaucoup de monde s'est retrouvé dans le cas de figure de reprendre du code d'une autre personne, d'y faire des modifications sans savoir si cela impactait le code d'origine. Dans ces cas là une politique de tests unitaires est très utile.

Mais c'est quoi concrètement des tests unitaires ?

Les tests unitaires sont un procédé permettant de s'assurer du fonctionnement correct d'une partie déterminée d'un logiciel ou d'une portion d'un programme (appelée « unité ») (Wikipédia).

Le principe de fonctionnement d'un test unitaire est d'isoler la portion de code à tester du reste du programme, et de tester la partie de code dans un environnement prédéterminé et de voir si cela réagis comme on le souhaite.

Les intérêts des tests unitaires sont multiples, nous pouvons citer par exemple le fait de tester chaque fonction indépendamment l'une de l'autre, mais aussi de garder à chaque instant un code fonctionnel et éviter la régression.

Comment intégrer des tests unitaires en C# ?

Il existe plusieurs solutions. Une solution qui est compatible avec tous les éditeurs Visual Studio ainsi que les différentes versions du Framework est Nunit.
Nunit est un Framework de tests unitaires : http://www.nunit.org

Dans les Visual studio Team system, un système de tests unitaires est intégré, mais depuis la version 2008 de Visual studio Professionnel, un tel système a également été intégré.
Dans ce tutorial c'est cette dernière option que nous allons traiter.

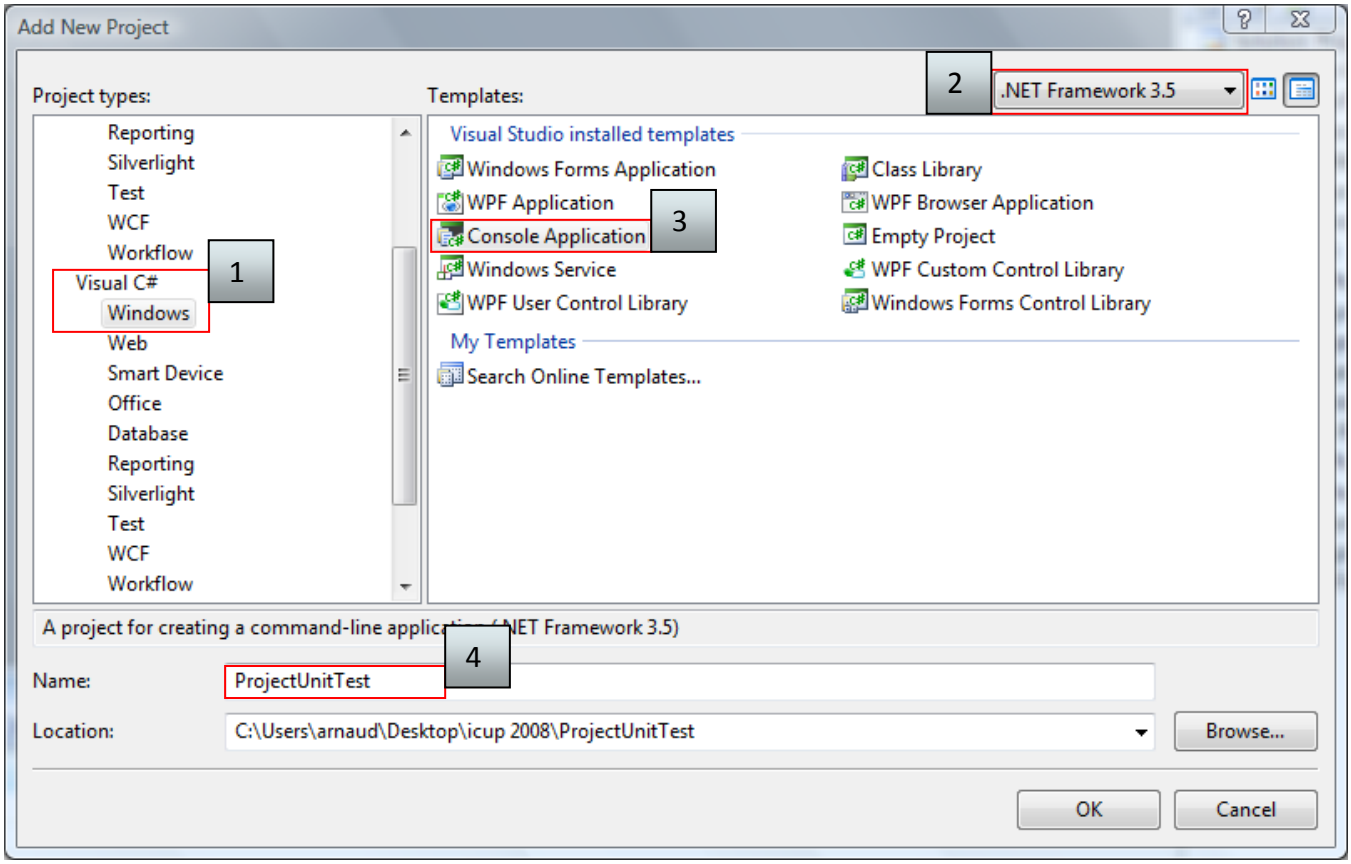
Avant de rentrer dans le vif du sujet nous allons créer, dans un premier temps un projet sur lequel nous appliquerons les tests unitaires. Nous nous attacherons également à la présentation des différentes interfaces.

Pré-requis

Nous allons commencer par créer dans un premier temps un projet nous permettant d'exécuter des tests unitaires dessus.

Pour cela nous allons créer un nouveau projet dans Visual Studio 2008, le projet sera du type application console en .Net 3.5. Le nom du projet sera :

ProjectUnitTest



Maintenant nous avons un projet créé. Nous allons lui ajouter un dossier pour la couche Dynamic Business Object qui contiendra notre classe de donnée « Person ». Puis un dossier pour la couche d'accès aux données, nommé DataAccessLayer, qui contiendra lui-même une classe « Person ».

Normalement la solution doit ressembler à cela maintenant :

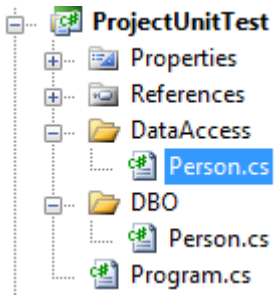


Figure 1 : Architecture du projet

Dans la classe Person du dossier DBO, nous allons mettre le code suivant :

```
public class Person
{
    #region variables
    /// <summary>
    /// nom de la personne
    /// </summary>
    private string _name;
    /// <summary>
    /// prénom de la personne
    /// </summary>
    private string _firstname;
    /// <summary>
    /// adresse de la personne
    /// </summary>
    private string _address;
    /// <summary>
    /// fonction de la personne
    /// </summary>
    private string _function;
    #endregion

    #region getter / setter
    /// <summary>
    /// accès à la focntion de la personne
    /// </summary>
    public string Function
    {
        get { return _function; }
        set { _function = value; }
    }
    /// <summary>
    /// accès au prénom de la personne
    /// </summary>
    public string Firstname
    {
        get { return _firstname; }
        set { _firstname = value; }
    }

    /// <summary>
    /// accès à l'adresse de la personne
    /// </summary>
    public string Address
    {
        get { return _address; }
        set { _address = value; }
    }

    /// <summary>
    /// accès au nom de la personne
    /// </summary>
    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    #endregion
    /// <summary>
    /// constructeur de la classe
    /// </summary>
    public Person()
    {
        _name = "";
        _firstname = "";
        _address = "";
        _function = "";
    }
```

```
        /// <summary>
        /// fonction overridée pour convertir l'objet personne sous forme d'une
        /// chaine de caractére
        /// </summary>
        /// <returns>l'objet sous forme d'une chaine de caractére</returns>
        public override string ToString()
        {
            return _name + " " + _firstname;
        }

        /// <summary>
        /// permet de tester la validité d'une date d'anniversaire pour 1 personne
        /// </summary>
        /// <param name="birthday">date d'anniversaire</param>
        /// <returns>vrai si la date est < à celle actuelle sinon faux</returns>
        public bool IsValidBirthday(DateTime birthday)
        {
            if (birthday < DateTime.Now)
            {
                return true;
            }
            else
            {
                return false;
            }
        }

    }
```

Pour des raisons de simplicité, nous n'allons pas suivre totalement la logique du développement en couche c'est pour cela que nous avons intégré la fonction IsValidBirthday dans la classe Person. En toute logique cette fonction aurait due être dans une autre classe. De la même façon pour éviter d'alourdir le projet nous ne mettrons pas de code dans la couche d'accès aux données.

Compiler le projet normalement, il n'y a pas d'erreurs. Dans la partie suivante nous allons procéder à la création du projet de test.

Création du projet de test.

Pour créer un projet de test il y a plusieurs solutions sur Visual Studio 2008.

On peut créer ce projet directement grâce à l'interface de test ou par l'ajout d'un projet dans une solution.

Présentation du menu

Astuce
Si l'interface de test n'apparait pas il faut faire un clic droit sur la barre de menu et cocher le menu Test Tools.

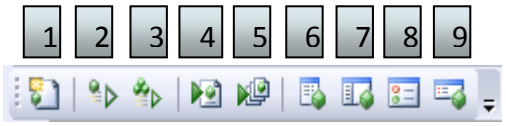


Figure 2 : Barre de test

Numéros	Description
1	Permet de créer un projet de test
2	Démarre les tests dans le projet courant
3	Démarre les tests pour toute la solution
4	Démarre les tests en mode « debug » dans le projet courant
5	Démarre les tests en mode « debug » pour toute la solution
6	Affiche la fenêtre de sélection des tests
7	Affiche l'éditeur de liste de tests
8	Affiche le résultat des tests
9	Affiche la fenêtre pour surveiller le système de test (tests en cours, tests en attente …)

Création du projet de test

Pour cela il y a deux méthodes :

- Par la barre de menu :
  Il faut cliquer sur le premier bouton du menu présenté ci-dessus. 1
  On accède ensuite à l'interface de sélection.
- Par l'ajout dans la solution :
  Effectuer un clic droit sur le nom de la solution dans l'explorateur de solution.
  Ajout d'un nouveau projet

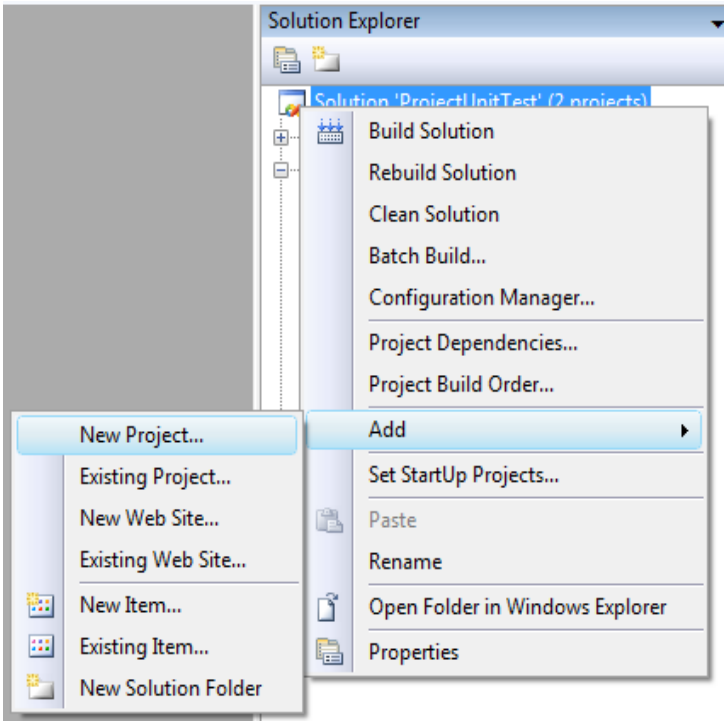


Figure 3 : Explorateur de la solution

Puis sélectionner un projet de test, que l'on nommera « TestProjectUnitTest » :

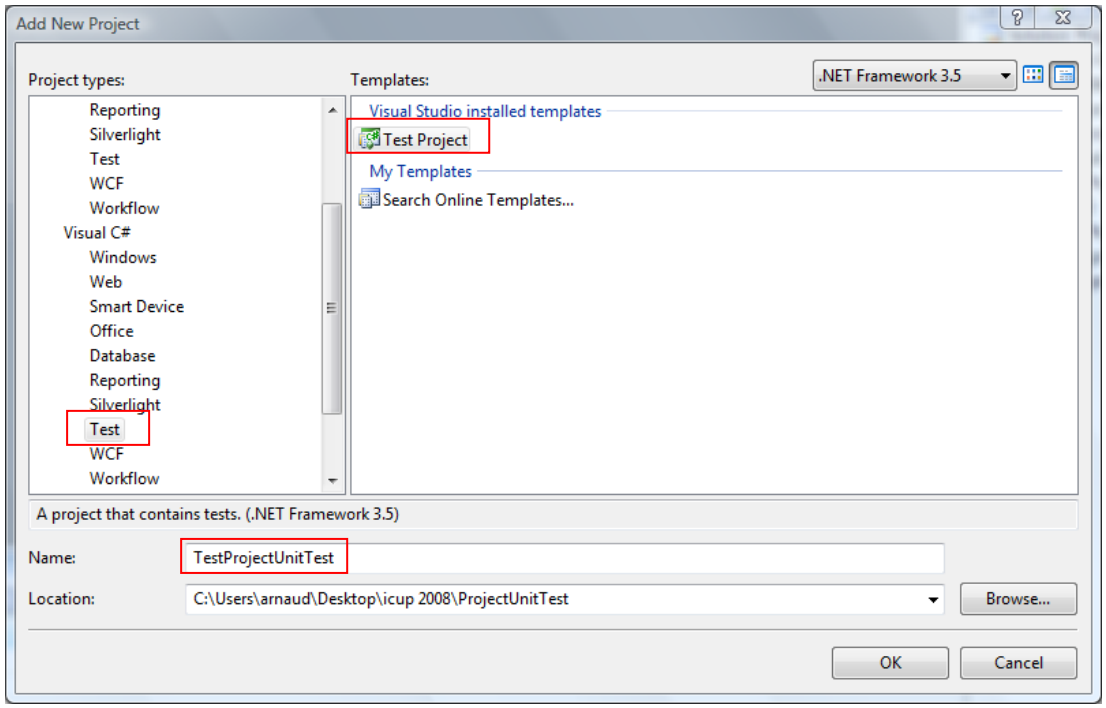


Figure 4 : Création du projet de test

Un nouveau projet est alors créé dans l'arborescence de la solution.

Création des tests

Pour ajouter des tests au projet de test, réaliser un clic droit dur TestProjectUnitTest :

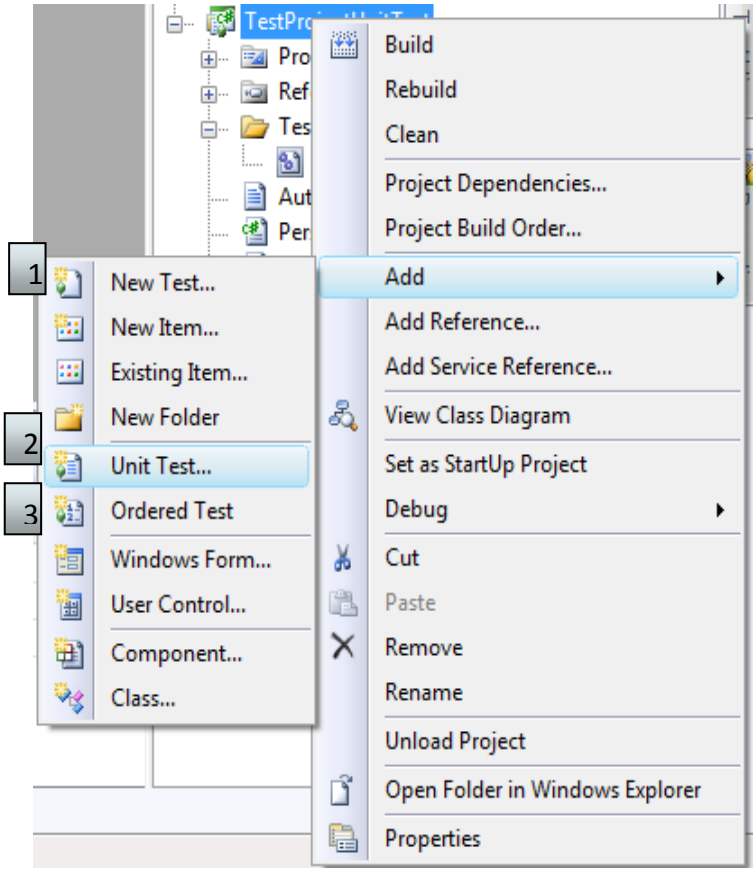


Figure 5 : Ajout de tests

Numéros	Description
1	Crée un fichier de test simple
2	Génère des tests automatiquement à partir de classe
3	Permet de générer un ordonnancement des tests

Dans notre cas, nous allons ajouter un fichier de type « Unit Test … », une nouvelle fenêtre apparait :

Pour générer les tests automatiquement, Visual Studio 2008 propose de sélectionner les classes sur lesquelles, on veut générer les tests. Puis cliquer sur OK.

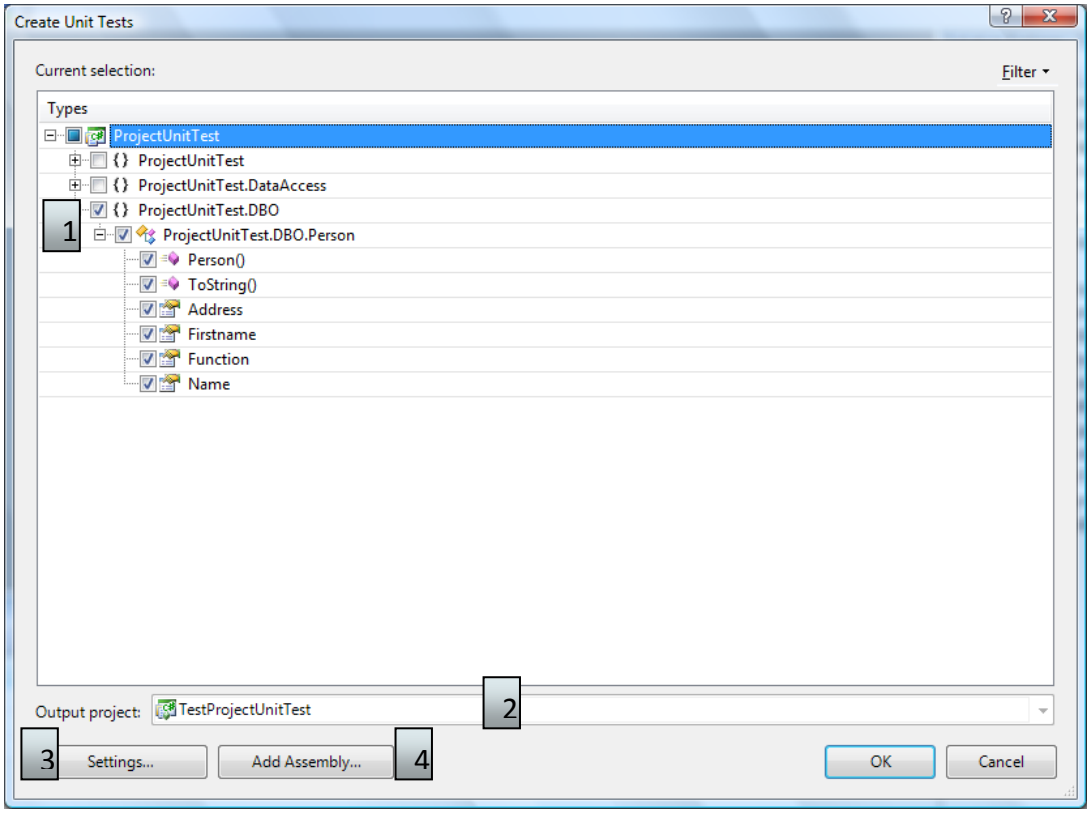


Figure 6 : Création des tests

Numéros	Description
1	Case pour sélectionner les classes à tester
2	Sélection du projet de test lorsque l'on a plusieurs projets de test
3	Permet de configurer la façon dont seront nommées les méthodes, …
4	Permet d'ajouter des assembly si celle-ci ne sont pas présentes par défaut dans la zone « Types »

Visual Studio 2008 génère les tests, et affiche la classe générée.

Pour voir les tests disponibles dans le projet on peut utiliser le visualisateur de tests.

Pour cela cliquer sur l'icône 1

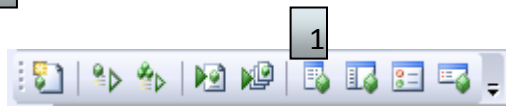


Figure 7 : bar de tests

Une nouvelle fenêtre s'affiche :

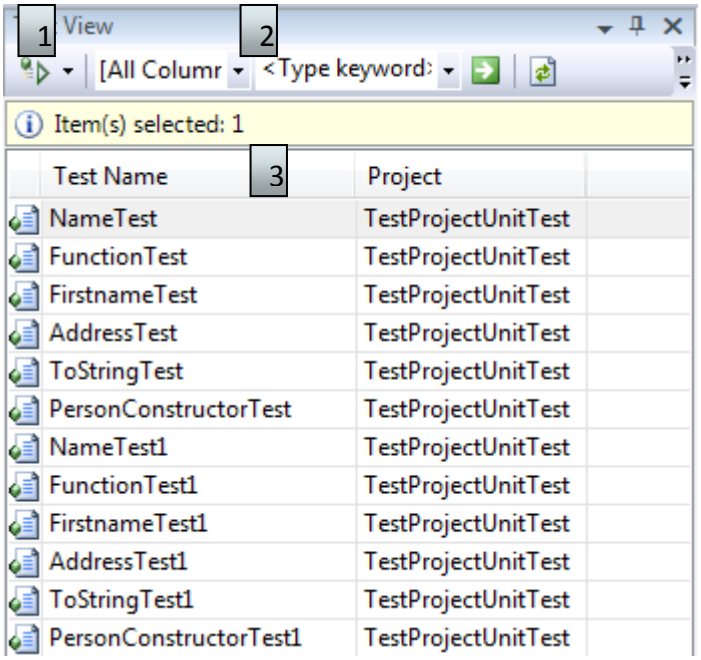


Figure 8 : Visualisation des tests

Numéros	Description
1	Permet de lancer les tests sélectionnés dans la liste
2	Permet de filtrer les tests affichés dans la liste
3	Affichage des tests

Maintenant que les tests sont générés, il faut les compléter. Pour aller directement sur un test, on peut se servir du visualisateur. Il suffit de faire un double clic sur le nom du test. Dans notre cas nous allons commencer par « NameTest ». Modifier la fonction pour réaliser le test. Le code suivant montre un exemple :

```
/// <summary>
///A test for Name
///</summary>
[TestMethod()]
[Owner("arnaud lemettre")]
public void NameTest()
{
  Person target = new Person(); // TODO: Initialize to an appropriate value
  string expected = "lemettre"; // TODO: Initialize to an appropriate value
  string actual;
  target.Name = expected;
  actual = target.Name;
  Assert.AreEqual(expected, actual);
}
```

Dans cette fonction on voit que Visual Studio instancie la classe Person. La variable « expected » correspond au résultat attendu par la fonctionnalité testée. Nous avons rajouté l'attribut « Owner » au dessus de la fonction. Cet attribut permet de spécifier l'auteur du test, cela peut être utile sur un projet où plusieurs personnes travaillent ensemble.

Le test se réalise grâce à la classe Assert. Dans notre cas la fonction utilisée est une fonction d'égalité. Bien entendu les tests ne s'arrêtent pas seulement à un test d'égalité, la classe fournie d'autres méthodes.

Nom	Description
AreEqual	Permet de tester l'égalité des deux paramètres
AreNotEqual	Permet de tester la différence entre les deux paramètres
AreSame	Permet de tester l'égalité sur les objets
AreNotSame	Permet de tester la différence sur les objets
Fail	Fait échouer automatiquement les tests
Inconclusive	Permet d'indiquer qu'un test n'est pas encore implémenté
IsFalse	Vérifie que la condition passée en paramètre est fausse
IsTrue	Vérifie que la condition passée en paramètre est vrai
IsInstanceOfType	Vérifie que l'objet passé en paramètre est bien du bon type
IsNotInstanceOfType	Vérifie que l'objet passé en paramètre n'est pas du type passé en paramètre
IsNull	Vérifie que l'objet passé en paramètre est null
IsNotNull	Vérifie que l'objet passé en paramètre n'est pas null

La page suivante vous fournira beaucoup plus d'informations détaillées :

http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assert.aspx

Il ne reste plus qu'à compléter le reste des tests.

Création de liste de tests

Pour avoir une politique de tests efficace il faut pouvoir les organiser pour cela Visual Studio propose un système de liste pour regrouper les tests. Pour accéder à cette fonctionnalité cliquer sur le bouton 1

Figure 9 : bar de tests

La fenêtre d'édition de liste apparait donc :

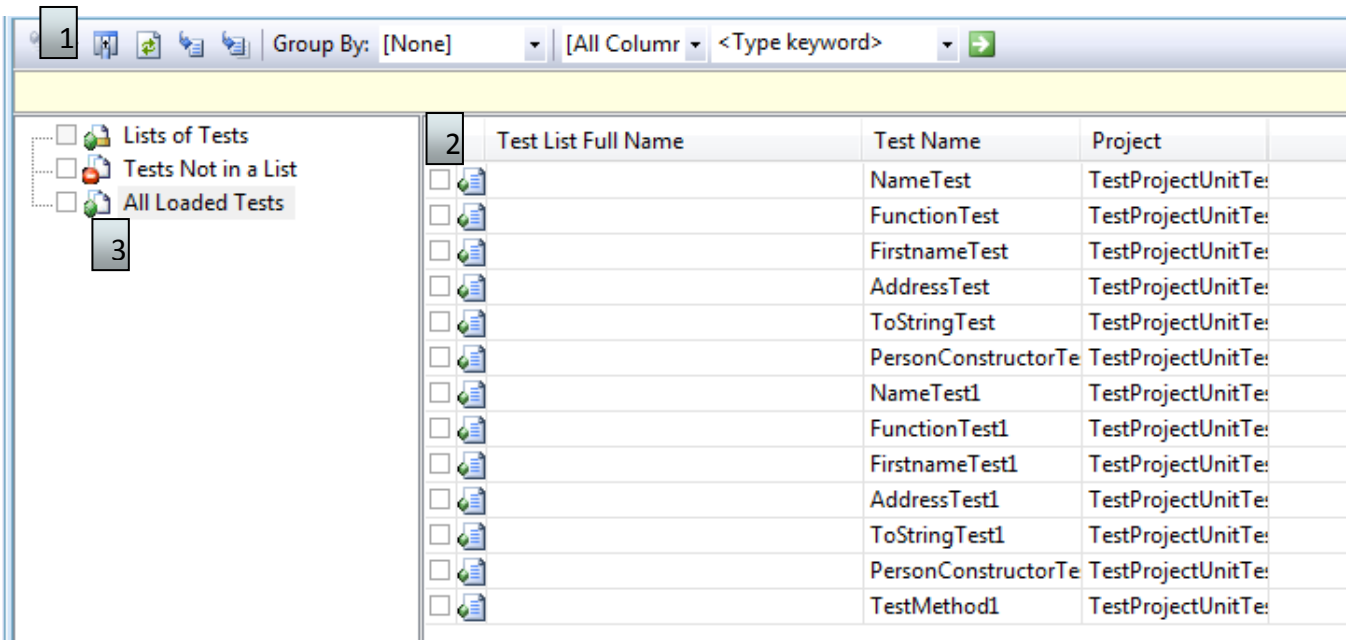


Figure 10 : Editeur de liste de tests

Numéros	Description
1	Plusieurs fonctionnalités permettant d'actualiser l'interface et de lancer des tests
2	Liste des tests. Permet de sélectionner les tests à exécuter.
3	Affichage des listes de tests

Cette interface à plusieurs utilités, la première est donc de créer des listes de tests, une des autres fonctionnalités est de pouvoir sélectionner les tests à exécuter.

Pour créer une liste de test faire un clic droit sur « Lists of tests »

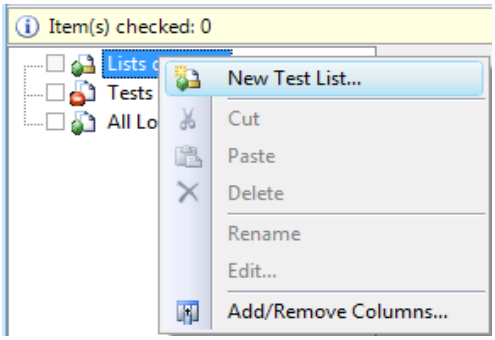


Figure 11 : Nouvelle liste de tests

Puis sélectionner « New Test List … »

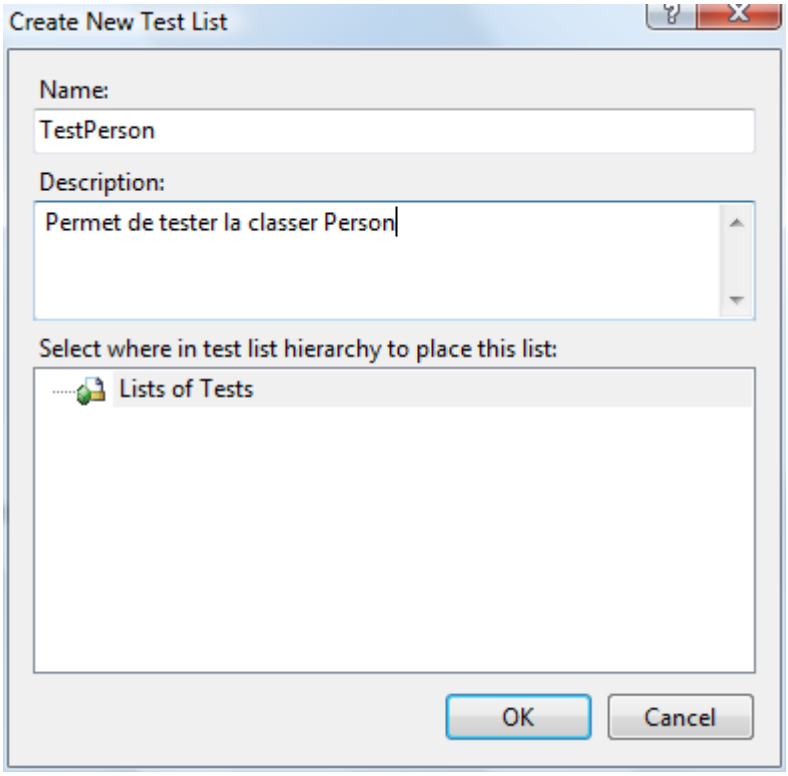


Figure 12 : Création d'une liste

Il faut remplir « Name » avec le nom de la liste que l'on veut créer, puis rentrer une description de la liste.

On peut par la suite effectuer une hiérarchie de tests.

Une fois rempli les informations cliquer sur OK.

Maintenant que la liste est créée il ne reste plus qu'à glisser-déposer les tests que l'on veut associer à cette liste. Pour cela cliquer sur « Tests Not in a List » sélectionner les tests à l'aide de la souris pour sélectionner plusieurs tests d'un coup utiliser la touche « shift » du clavier, et glisser les tests sur la liste que nous venons de créer.

Astuce
Pour créer une liste de test on peut directement, sélectionner les tests dans l'interface, réaliser un clic droit dessus et faire « New Test List … », les tests seront alors automatiquement ajoutés à la liste.

Exécution d'un test

Les tests peuvent se lancer de plusieurs endroits sous Visual Studio. On peut les lancer du visualisateur de tests, mais aussi de l'éditeur de liste de tests. Nous allons ici les lancer à partir de l'éditeur de tests.

Pour lancer un test, sélectionner dans l'arborescence le nœud où se trouve le test, soit il est dans une liste ou alors on peut également y accéder à partir de la liste « All Loaded Tests » qui regroupe tous les tests chargés du projet.

Nous allons tester pour notre exemple le Getter / Setter « Name » de la classe Person.

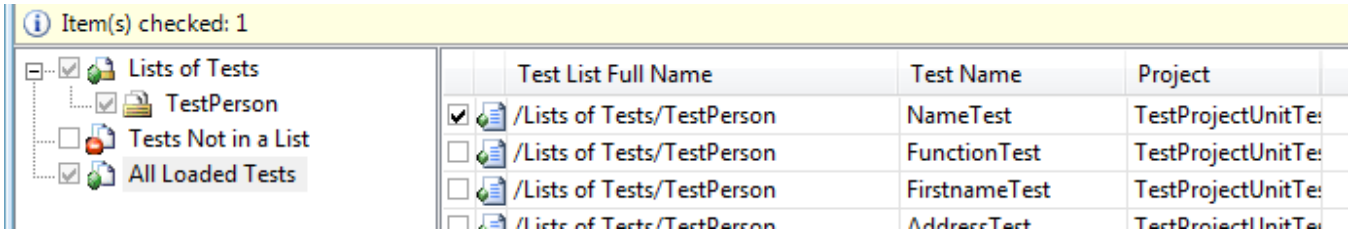


Figure 13 : Sélection d'un test

Pour cela il faut cliquer sur la case devant le test appelé « NameTest ».

Pour lancer le test il faut cliquer sur l'icône  qui se situe dans la barre juste au dessus de l'éditeur.

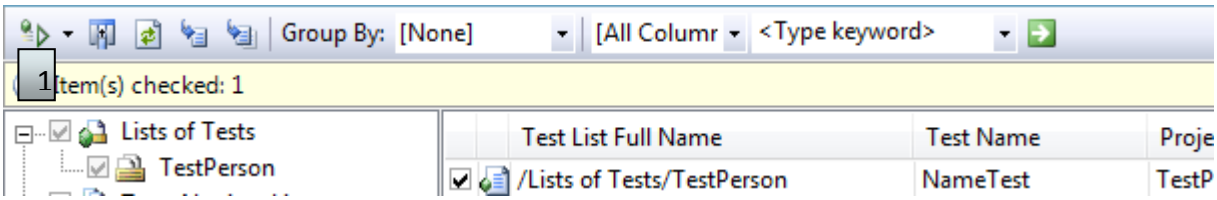


Figure 14 : Barre de l'éditeur

Ce bouton permet de lancer deux types de tests. Lance les tests en mode Debug cela permet de voir les erreurs directement dans le code. Et un mode normal qui exécute les tests et affichent les résultats. Par défaut les tests sont lancés uniquement dans un mode d'exécution et non de debug.

Quand les tests sont finis d'exécuter une fenêtre de résultat s'ouvre :

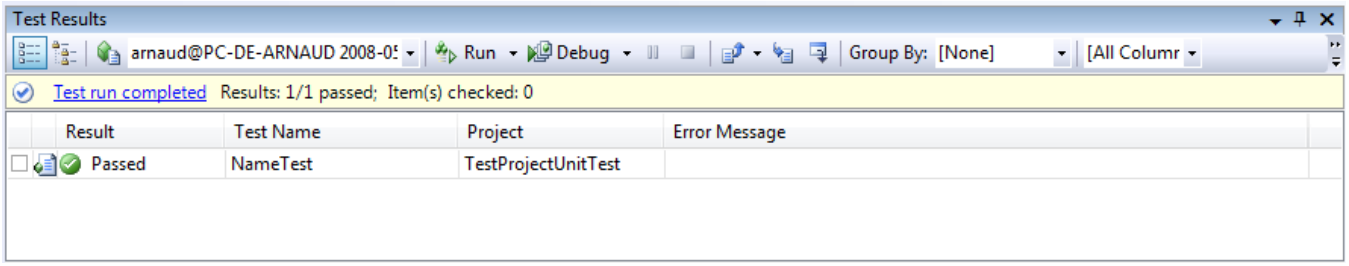


Figure 15 : Visualisation des résultats

La fenêtre de résultat indique les tests qui ont réussis et ceux qui ont échoué. Lorsqu'un test échoue le message d'erreur s'affiche. Ce message peut être personnalisé en surchargeant les méthodes appelées par la classe Assert lors de la création des tests, ceci afin de donner plus de détail.

Lancement d'une liste de tests

Nous pouvons également lancer une liste de tests cela fonctionne de la même manière qu'un simple test. Pour cela il suffit de cliquer devant le nom de la liste de tests que l'on veut exécuter pour la sélectionner. Bien entendu, on peut sélectionner plusieurs listes et également des tests qui ne sont pas dans une liste.

Tests Ordonnés

Lors de la création de vos tests il se peut que l'appel de certains tests se fait dans un ordre précis hors jusqu'à présent nous avons aucune maitrise sur l'ordre de lancement des tests. Cependant nous

n'avons pas encore utilisé les « Ordored Test » ce type d'item permet comme son nom l'indique de pouvoir ordonner ces tests.

Pour ajouter un ordre il faut réaliser un clic droit sur le projet de test, puis faire ajout Ordored Test.

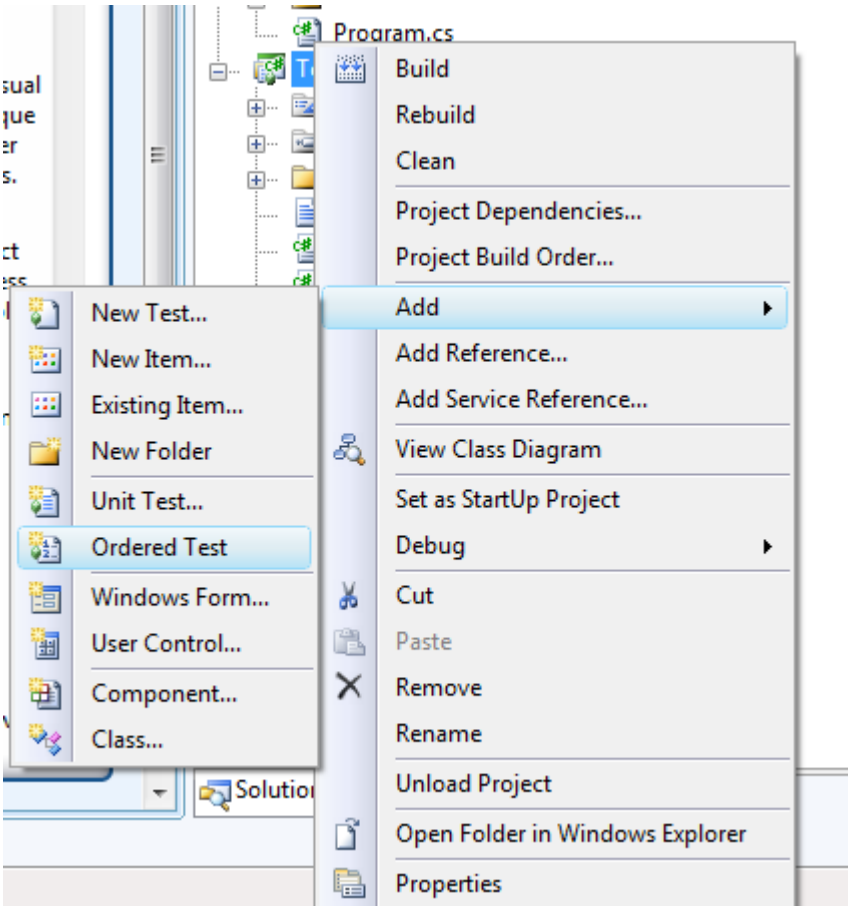


Figure 16 : ajout test ordonné

Une interface s'ouvre permettant de lister l'ensemble des tests unitaires du projet. Pour les mettre dans la liste il faut sélectionner tous les tests que vous souhaitez ordonner puis cliquer sur la flèche pour les transférer dans la partie droite. A partir de ce moment il ne reste plus qu'à jouer avec les flèches pour faire monter ou descendre les tests, pour spécifier l'ordre.

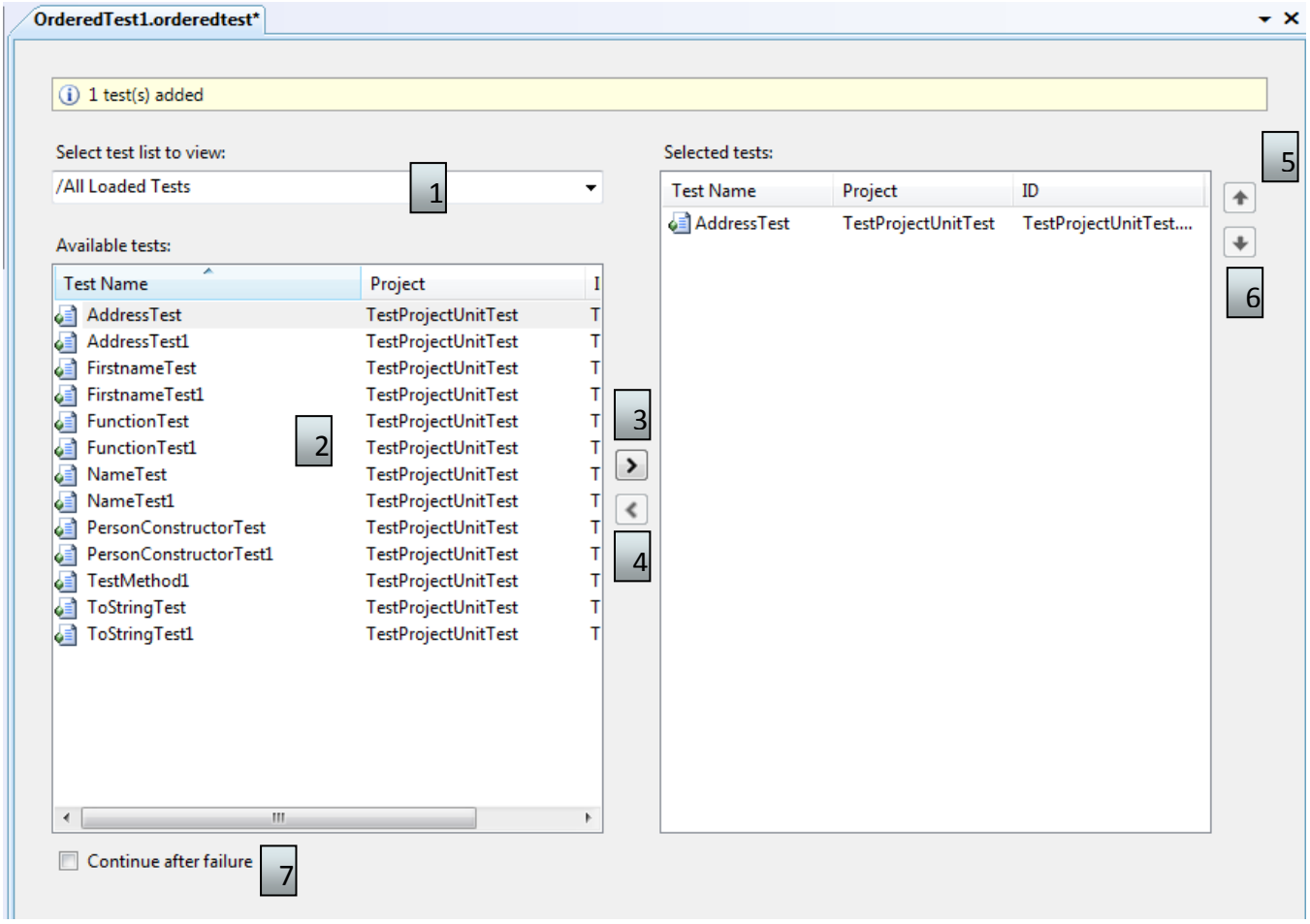


Figure 17 : Interface des tests ordonnées

Numéros	Description
1	Permet d'affiner l'affichage des tests en choisissant directement dans les listes.
2	Liste des tests.
3	Permet d'ajouter des tests.
4	Permet de retirer les tests.
5	Permet de faire monter dans l'ordre le test.
6	Permet de faire descendre dans l'ordre le test.
7	Indique si on veut continuer ou pas si un test échoue dans l'ordre spécifié.

Pour prendre en compte votre ordre vous devez sauvegarder, pour cela faite un :

```
Ctrl + s
```

Exécution de la liste de tests ordonnés

Retourner dans l'éditeur de la liste de tests. Le test ordonné apparait maintenant dans la liste. Vous pouvez le déplacer dans une liste existante ou en créer une nouvelle.

Pour lancer le test cocher le test. Les tests que contient ce test ordonné ne doivent pas être cochés, sinon les tests seront exécutés 2 fois.

Pour le lancer faite comme précédemment et le résultat s'affichera en bas.

Liaison avec des fichiers

Tester une fonctionnalité c'est bien mais le fait de faire qu'un seul test n'est pas assez fiable. Une solution serait de faire du copier coller des tests et de changer juste les valeurs.

Pour nous simplifier la vie, une option permet de spécifier une connexion à une base de données ou un fichier XML par exemple, afin de fournir un jeu de données de tests à l'application.

Dans notre projet, par exemple imaginons que nous avons une fonction qui test si la date de naissance est valide (inférieur à la date d'aujourd'hui).

Pour cela créer un nouveau fichier XML :

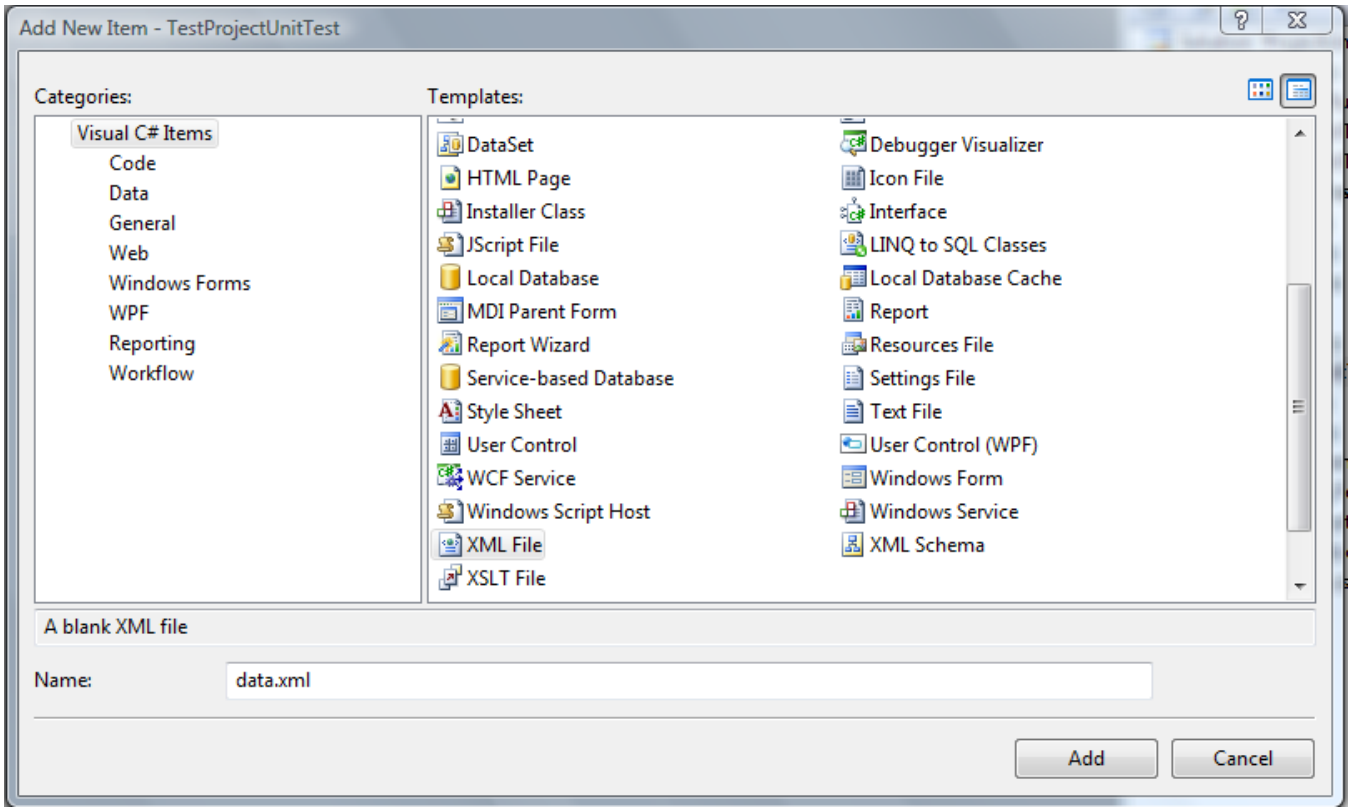


Figure 18 : ajout fichier XML

Clic droit sur le projet de test, puis ajouter un nouvel item. Sélectionner fichier XML. Nommez le data.xml, cliquer sur Add et copier coller les valeurs si dessous.

```
<?xml version="1.0" encoding="utf-8" ?>
<values>
  <person>
    <name>titi</name>
    <birthday>20/12/2005</birthday>
  </person>
  <person>
    <name>toto</name>
    <birthday>20/12/2009</birthday>
  </person>
</values>
```

Ce fichier nous servira de données de test, il faut maintenant spécifier au test qu'il faut qu'il utilise ce fichier. Pour cela dans la partie « test view » effectuer un clic droit sur le test IsValidBirthdayTest pour afficher les propriétés.

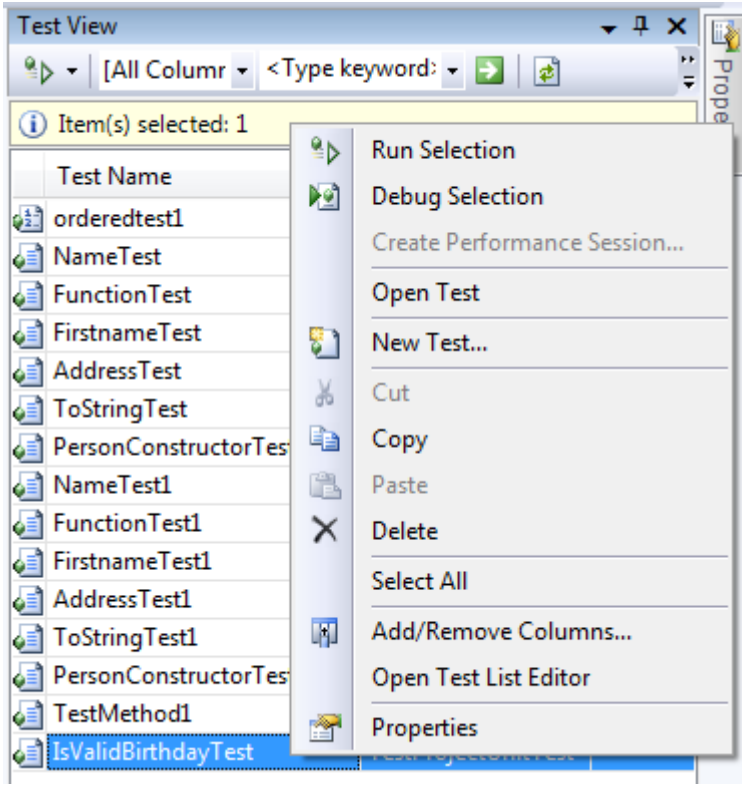


Figure 19 : Accès aux propriétés

Dans la partie propriété cliquer sur les « … » dans la propriété « Data Connection String »

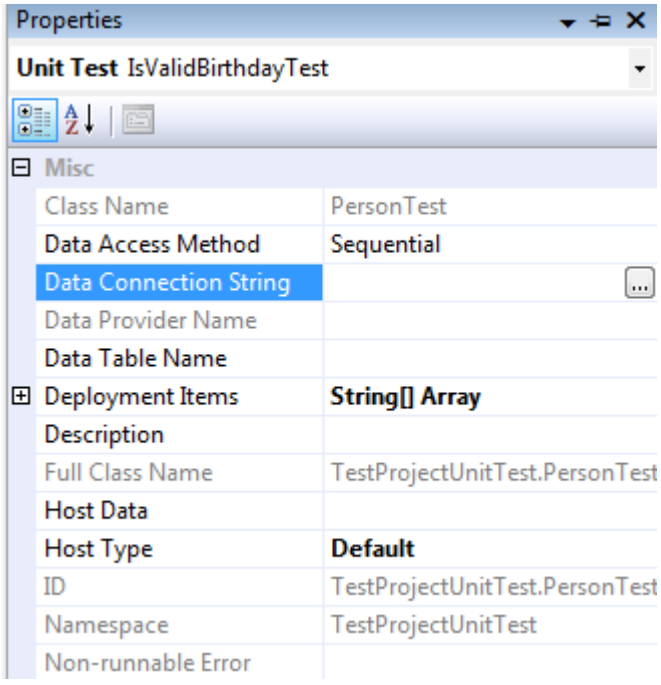


Figure 20 : Propriété chaine de connexion

Une nouvelle apparait, dans notre cas nous allons sélectionner le fichier XML, mais si les données sont en base, il faut juste sélectionner base de données.

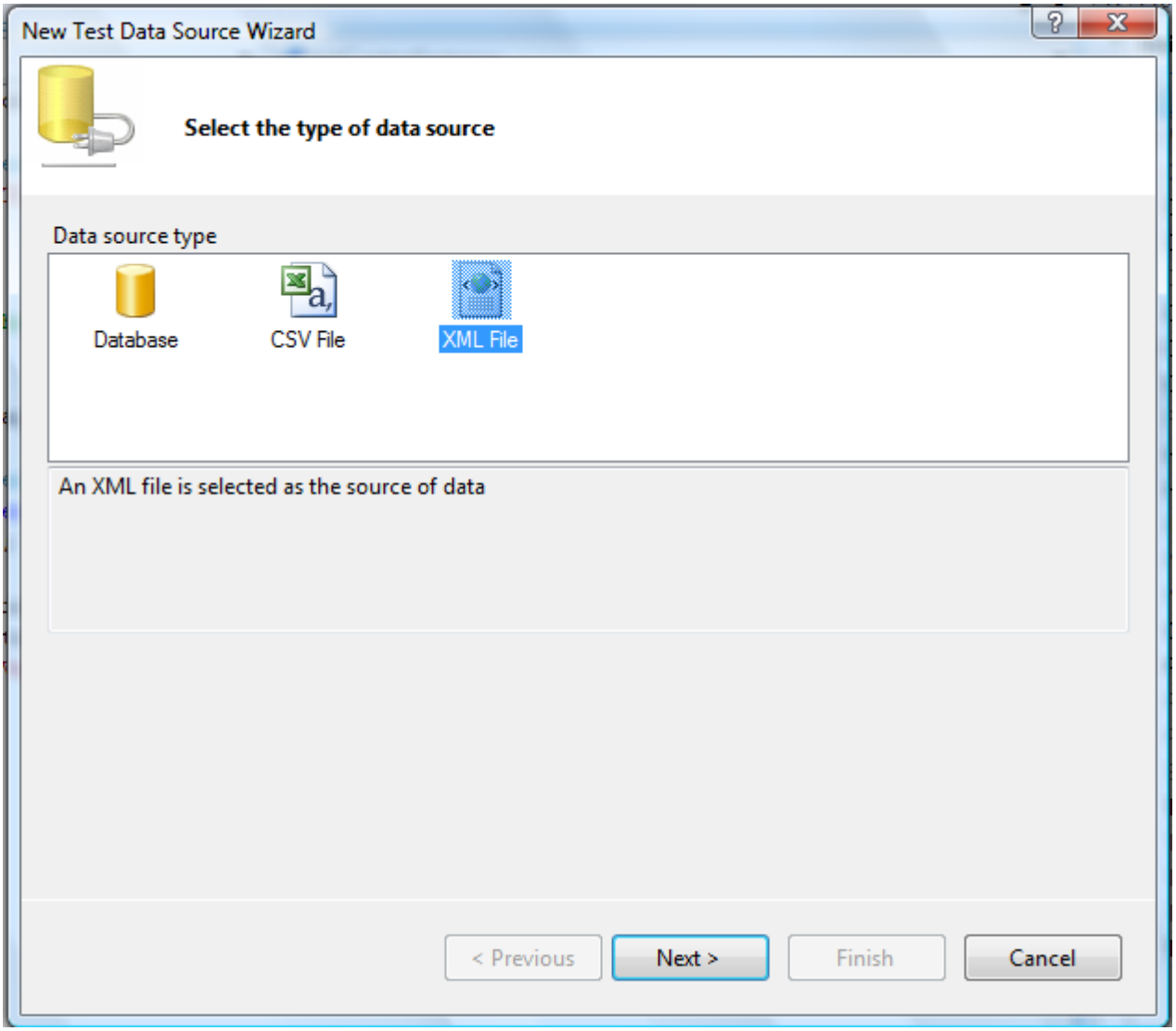


Figure 21 : Sélection du fichier XML

Il reste à spécifier le chemin d'accès du fichier et sélectionner le nœud dans notre cas « person », puis de cliquer sur next et finish.

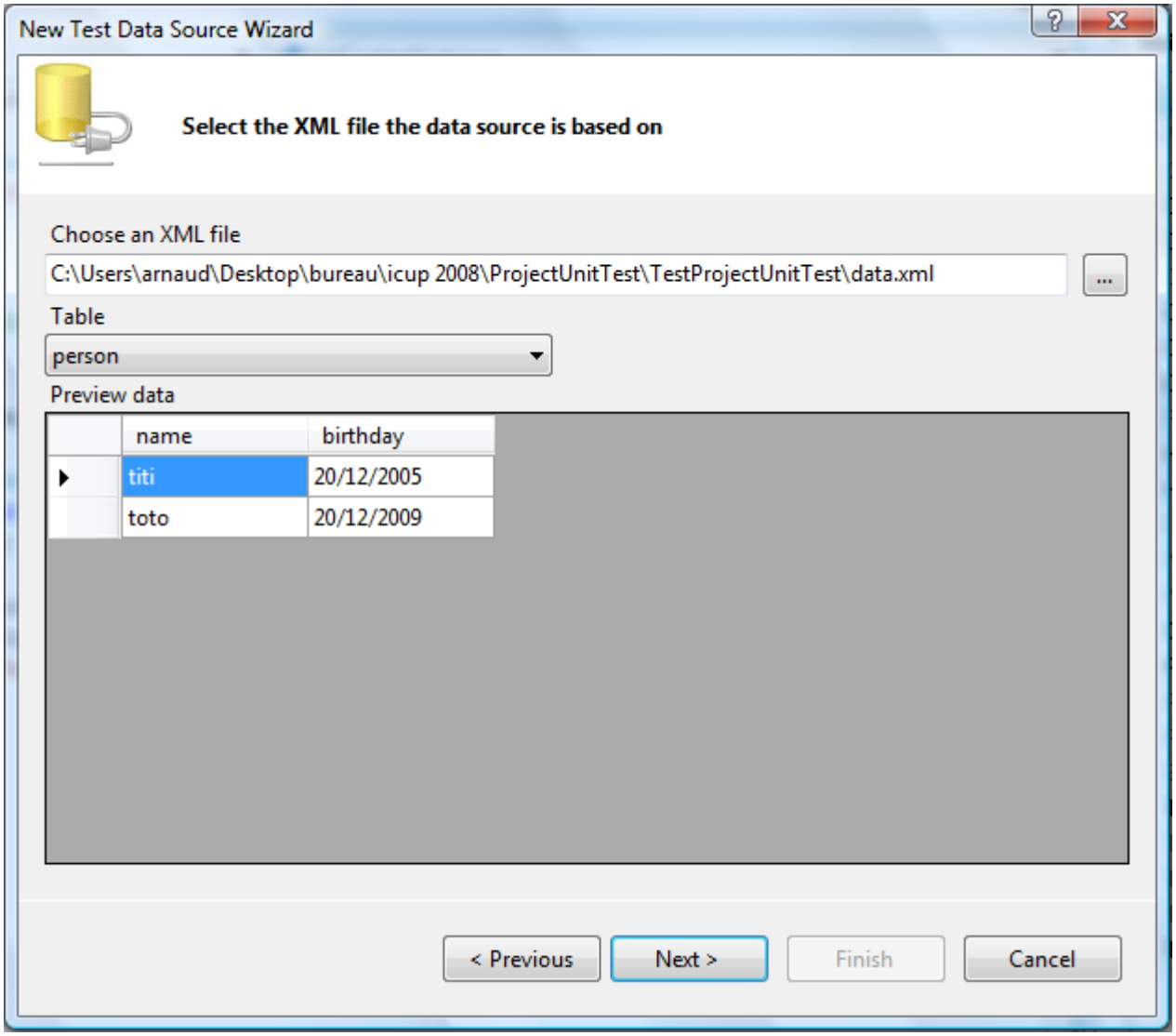


Figure 22 : Sélection des données

L'assistant vient alors de rajouter des informations au dessus de la méthode de test. Une des options intéressantes et de mettre l'exécution des données de manière aléatoire ou séquentiel, en le spécifiant dans la propriété « Data Access Method ».

Cependant il reste encore le fait de spécifier dans la méthode de test les valeurs à utiliser. Pour cela il faut utiliser le contexte de test.

```
DateTime birthday = Convert.ToDateTime(TestContext.DataRow["birthday"]);
```

Le contexte de test est accessible grâce à « `TestContext` ». La propriété DataRow permet d'accéder aux différents nœuds.

Le lancement du test se fait comme auparavant.

Conclusion

Dans cet article, nous avons vu l'utilisation de tests unitaires dans Visual Studio, c'est une partie de la programmation extrêmement importante notamment lors de développement à plusieurs. De plus cela évite la régression du logiciel au fur et à mesure des évolutions. Visual Studio permet de nous accompagner dans cette phase, d'une façon quasiment transparente. Nous avons fait un rapide tour des fonctionnalités que propose Visual studio il y a encore de nombreux scénarios à exploiter.