eBook Gratuit

APPRENEZ nunit

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#nunit

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: nunit

It is an unofficial and free nunit ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official nunit.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec nunit

Remarques

Cette section fournit une vue d'ensemble de ce qu'est une unité et pourquoi un développeur peut vouloir l'utiliser.

Il convient également de mentionner tous les sujets importants dans la police et d'établir un lien avec les sujets connexes. La documentation de nunit étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Versions

Version	Date de sortie
2.2	2004-08-08
2.2.1	2004-10-26
2.2.2	2004-12-07
2.2.3	2005-02-14
2.2.4	2005-12-14
2.2.5	2005-12-22
2.2.6	2006-01-21
2.2.7	2006-02-18
2.2.8	2006-04-21
2.2.9	2006-11-26
2.2.10	2007-03-15
2.4 RC1	2007-02-25
2.4 (version finale)	2007-03-16
2.4.1	2007-05-03
2.4.2	2007-08-02
2.4.4	2007-10-20
2.4.5	2007-11-23

Version	Date de sortie
2.4.6	2007-12-31
2.4.7	2008-03-30
2.4.8	2008-07-21
2,5	2009-05-02
2.5.1	2009-07-08
2.5.2	2009-08-10
2.5.3	2009-12-11
2.5.4	2010-04-08
2.5.5	2010-04-22
2.5.6	2010-07-24
2.5.7	2010-08-01
2.5.8	2010-10-14
2.5.9	2010-12-14
2.5.10	2011-04-02
2.6	2012-02-20
2.6.1	2012-08-04
2.6.2	2012-10-22
2.6.3	2013-10-10
2.6.4	2014-12-16
3.0 (alpha 1)	2014-09-22
3.0 (bêta 1)	2015-03-25
3.0 RC1	2015-11-01
3.0.0 Version finale	2015-11-15
3.0.1	2015-12-01
3.2	2016-03-05

Version	Date de sortie
3.2.1	2016-04-19
3.4	2016-06-25

Examples

Installation à l'aide de NuGet

```
Install-Package NUnit
```

Ce package comprend tous les assemblages nécessaires à la création de tests unitaires.

Les tests peuvent être exécutés en utilisant l'une des méthodes suivantes:

- Fenêtre de test de l'unité Visual Studio
- Coureur de console
- Un tiers qui prend en charge NUnit 3

Fenêtre de test de l'unité Visual Studio

Pour exécuter des tests à l'aide de la fenêtre de test de l'unité Visual Studio, installez l'adaptateur de test NUnit 3. https://visualstudiogallery.msdn.microsoft.com/0da0f6bd-9bb6-4ae3-87a8-537788622f2d

Console Runner

Installez le programme d'exécution de la console NUnit via NuGet

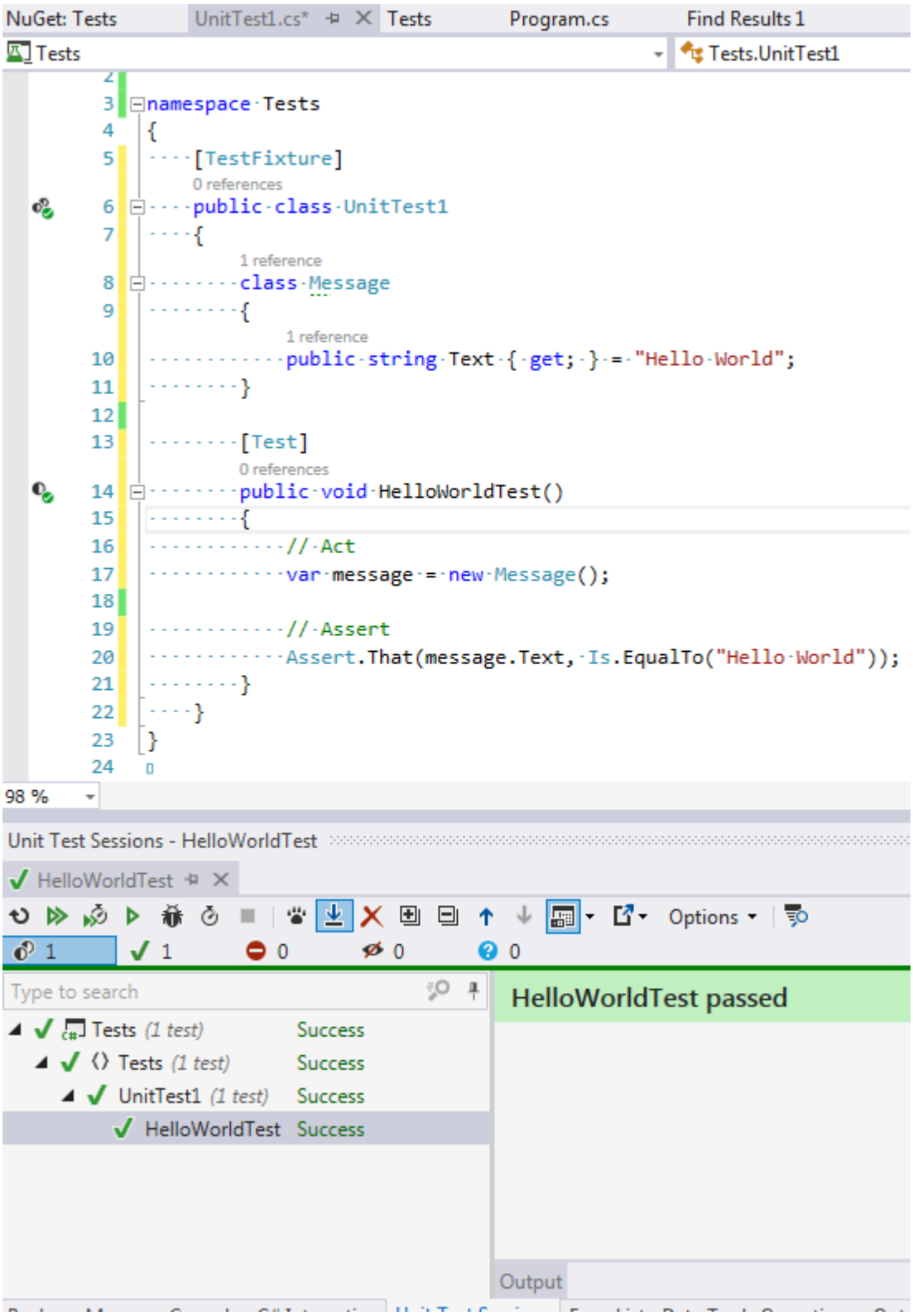
```
Install-Package NUnit.Console
```

L'exécutable nunit3-console.exe se trouve dans packages \ NUnit.3.XX \ tools

Bonjour le monde

```
[TestFixture]
public class UnitTest1
{
    class Message
    {
        public string Text { get; } = "Hello World";
    }

    [Test]
    public void HelloWorldTest()
```

```
    {
        // Act
        var message = new Message();

        // Assert
        Assert.That(message.Text, Is.EqualTo("Hello World"));
    }
}
```



Pourquoi vous ne pouvez pas utiliser Assert.Equals

Vous êtes-vous déjà demandé pourquoi vous ne pouvez pas utiliser Assert.Equals () pour Nunit et MSTest. Si vous ne l'avez pas encore peut-être, vous devez savoir que vous ne pouvez pas utiliser cette méthode. Au lieu de cela, vous utiliseriez Assert.AreEqual () pour comparer deux objets pour l'égalité.

La raison ici est très simple. Comme toute classe, la classe Assert hérite de System.Object qui a une méthode publique virtuelle Equals destinée à vérifier si un objet donné est égal à l'objet actuel. Par conséquent, appeler cette méthode égale serait une erreur, car dans un test unitaire, vous devriez comparer deux objets qui n'ont rien à voir avec la classe Assert. Nunit et MSTest ont donc choisi de fournir une méthode Assert.AreEqual à cette fin.

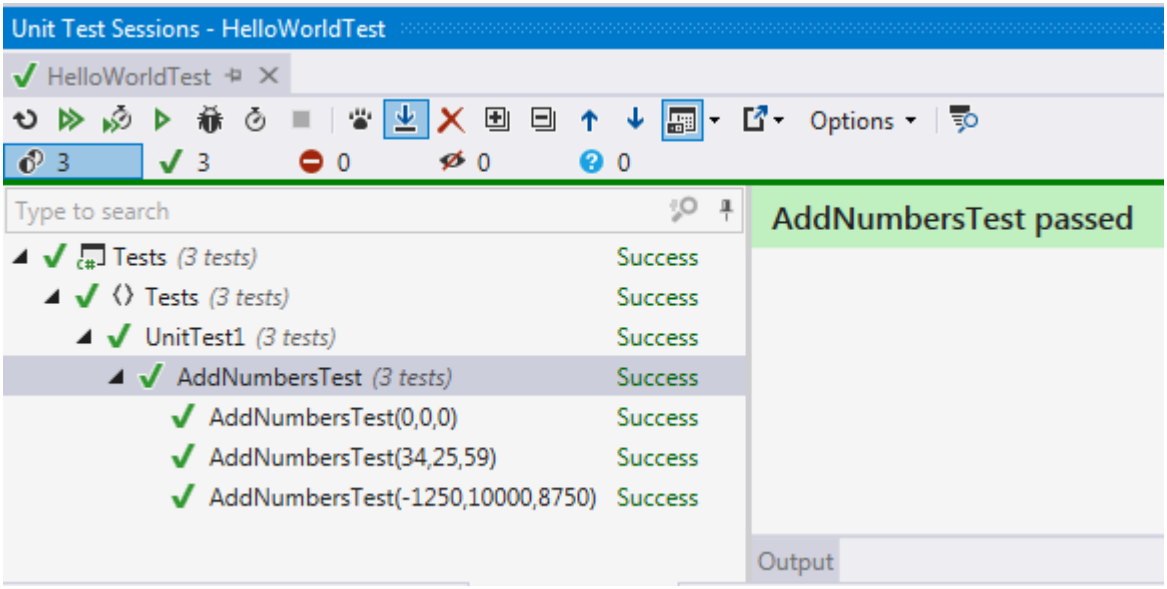
De plus, pour vous assurer de ne pas utiliser la méthode Equals par erreur, vous avez décidé de lancer des exceptions pour vous avertir si vous les utilisez par erreur.

Implémentation Nunit:

```
        [EditorBrowsable(EditorBrowsableState.Never)]
    public static new bool Equals(object a, object b)
    {
        // TODO: This should probably be InvalidOperationException
        throw new AssertionException("Assert.Equals should not be used for Assertions");
    }
```

TestCaseAttribute

```
[TestCase(0, 0, 0)]
[TestCase(34, 25, 59)]
[TestCase(-1250, 10000, 8750)]
public void AddNumbersTest(int a, int b, int expected)
{
    // Act
    int result = a + b;

    // Assert
    Assert.That(result, Is.EqualTo(expected));
}
```



Lire Commencer avec nunit en ligne: https://riptutorial.com/fr/nunit/topic/1738/commencer-avec-nunit

Chapitre 2: Assertions Courantes

Remarques

La forme `Assert.That()` de NUnit prend en charge l'utilisation de contraintes comme second paramètre. Toutes les contraintes fournies par NUnit sont disponibles via les classes statiques `Is` , `Has` et `Does` . Les contraintes peuvent être combinées en expressions fluides en utilisant les méthodes intégrées `And` , `Or` et `With` . Les expressions peuvent être facilement développées à l'aide des nombreuses méthodes de `ConstraintExpression` , telles que `AtMost` et `Contains` .

Examples

Affirmation de base

```
Assert.That(actual, Is.EqualTo(expected));
```

Utilisation avancée des contraintes

Les grandes assertions courantes deviennent plus difficiles à lire, mais lorsqu'elles sont combinées avec des classes dotées de bonnes implémentations de `ToString()` , elles peuvent générer des messages d'erreur très utiles.

```
[Test]
public void AdvancedContraintsGiveUsefulErrorMessages() {
    Assert.That(actualCollection, Has
        .Count.EqualTo(4)
        .And.Exactly(1).Property("Age").GreaterThan(60)
        .And.Some.Property("Address").Null
        .And.No.Property("Age").LessThanOrEqualTo(17));
}
```

En cas d'échec, cette assertion génère des messages comme ceux-ci:

```
Expected: property Count equal to 4 and exactly one item property Age greater
than 60 and some item property Address null and not property Age less than or
equal to 17
But was:  < <Steve Taylor (23) lives in Newcastle
>, <Michelle Yung (65) lives in San Francisco
>, <Ranjit Saraman (49) lives in Milano
>, <LaChelle Oppenheimer (16) lives in
> >
```

Collections

```
var a = new List<int> { 1, 2 };
var b = new List<int> { 2, 1 };
```

```
Assert.That (a, Is.EqualTo(b)); // fails
Assert.That (a, Is.EquivalentTo(b)); // succeeds
```

Lire Assertions Courantes en ligne: https://riptutorial.com/fr/nunit/topic/2788/assertions-courantes

Chapitre 3: Écrire une contrainte personnalisée pour le modèle de contrainte

Examples

Faire correspondre un entier approximativement

Supposons que nous voulions écrire une contrainte correspondant à un nombre, mais approximativement. Disons que vous êtes censé avoir `95` personnes dans une enquête, mais que `93` ou `96` feront également. Nous pouvons écrire une contrainte personnalisée de la forme:

```
public class AlmostEqualToConstraint : Constraint
{
    readonly int _expected;
    readonly double _expectedMin;
    readonly double _expectedMax;
    readonly int _percentageTolerance;


    public AlmostEqualToConstraint(int expected, int percentageTolerance)
    {
        _expected = expected;
        _expectedMin = expected * (1 - (double)percentageTolerance / 100);
        _expectedMax = expected * (1 + (double)percentageTolerance / 100);
        _percentageTolerance = percentageTolerance;
        Description = $"AlmostEqualTo {expected} with a tolerance of {percentageTolerance}%";
    }


    public override ConstraintResult ApplyTo<TActual>(TActual actual)
    {
        if (typeof(TActual) != typeof(int))
            return new ConstraintResult(this, actual, ConstraintStatus.Error);

        var actualInt = Convert.ToInt32(actual);

        if (_expectedMin <= actualInt && actualInt <= _expectedMax)
            return new ConstraintResult(this, actual, ConstraintStatus.Success);
        else
            return new ConstraintResult(this, actual, ConstraintStatus.Failure);
    }
}
```

Rendre une nouvelle contrainte utilisable dans un contexte fluide

Nous allons intégrer le `AlmostEqualToConstraint` avec les interfaces NUnit couramment, en particulier le `Is` . Nous devons étendre le NUnit fourni `Is` et utiliser que tout au long de notre code.

```
public class Is : NUnit.Framework.Is
{
    public static AlmostEqualToConstraint AlmostEqualTo(int expected, int percentageTolerance
```

```
= 5)
    {
        return new AlmostEqualToConstraint(expected, percentageTolerance);
    }
}
```

Lire Écrire une contrainte personnalisée pour le modèle de contrainte en ligne: https://riptutorial.com/fr/nunit/topic/9273/ecrire-une-contrainte-personnalisee-pour-le-modele-de-contrainte

Chapitre 4: Les attributs

Remarques

La version 1 de NUnit utilisait l'approche classique pour identifier les tests basés sur les conventions d'héritage et de nommage. A partir de la version 2.0, NUnit a utilisé des attributs personnalisés à cette fin.

Les montages de test NUnit n'héritant pas d'une classe de structure, le développeur est libre d'utiliser l'héritage d'une autre manière. Et comme il n'y a pas de convention arbitraire pour nommer les tests, le choix des noms peut être entièrement orienté vers la communication de l'objectif du test.

Tous les attributs NUnit sont contenus dans l'espace de noms NUnit.Framework. Chaque fichier source contenant des tests doit inclure une instruction using pour cet espace de noms et le projet doit faire référence à l'assembly de structure, nunit.framework.dll.

À partir de NUnit 2.4.6, les attributs de NUnit ne sont plus scellés et tous les attributs qui en dérivent seront reconnus par NUnit.

Examples

TestCaseAttributeExample

```
[TestCase(0, 0, 0)]
[TestCase(34, 25, 59)]
[TestCase(-1250, 10000, 8750)]
public void AddNumbersTest(int a, int b, int expected)
{
    // Act
    int result = a + b;

    // Assert
    Assert.That(result, Is.EqualTo(expected));
}
```

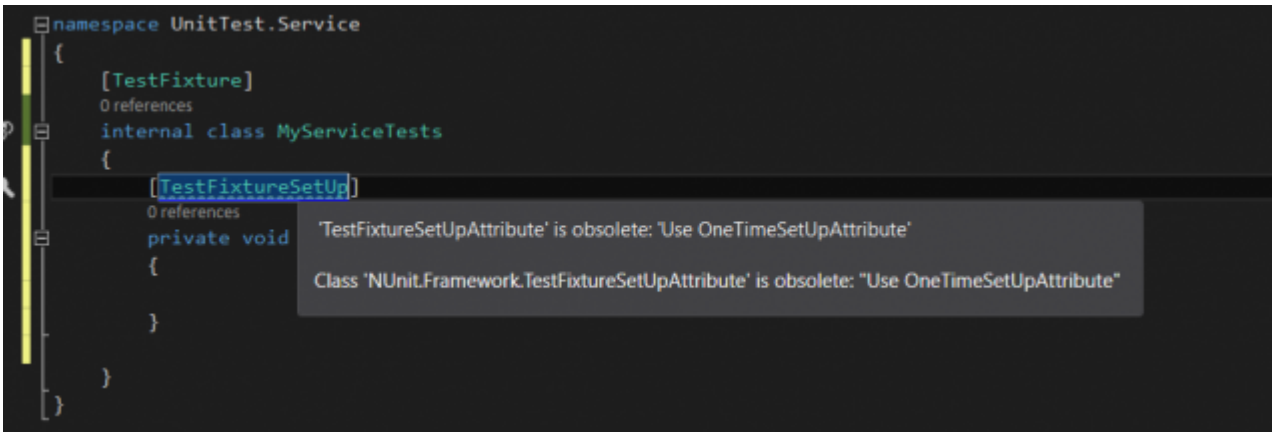
TestFixture

```
[TestFixture]
public class Tests {

    [Test]
    public void Test1() {
        Assert.That(true, Is.EqualTo(true));
    }

}
```

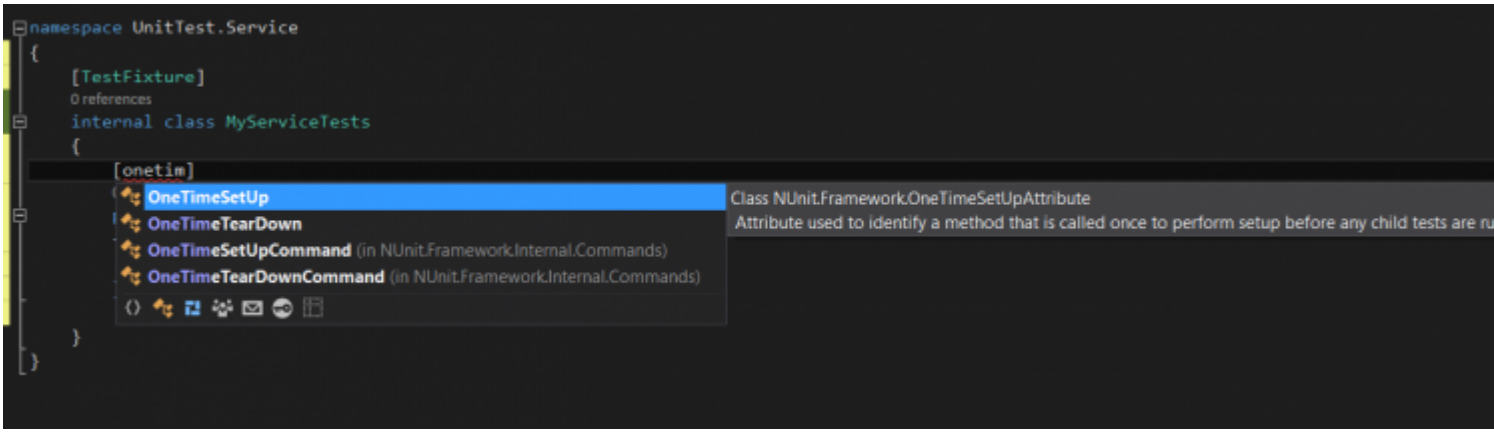
Un dispositif de test marque une classe comme contenant des tests.

TestFixtureSetUp

Cet attribut utilisé pour identifier une méthode appelée une fois pour effectuer la configuration avant tout test enfant est exécuté. Pour les nouvelles versions, nous utilisons *OneTimeSetUp* car TestFixtureSetUp est obsolète.



OneTimeSetUp




```
[OneTimeSetUp]
public void SetUp()
 {

 }
```

Abattre

Cet attribut est utilisé pour identifier une méthode qui est appelée immédiatement après chaque test, elle sera appelée même s'il y a une erreur, c'est l'endroit où nous pouvons disposer de nos objets.

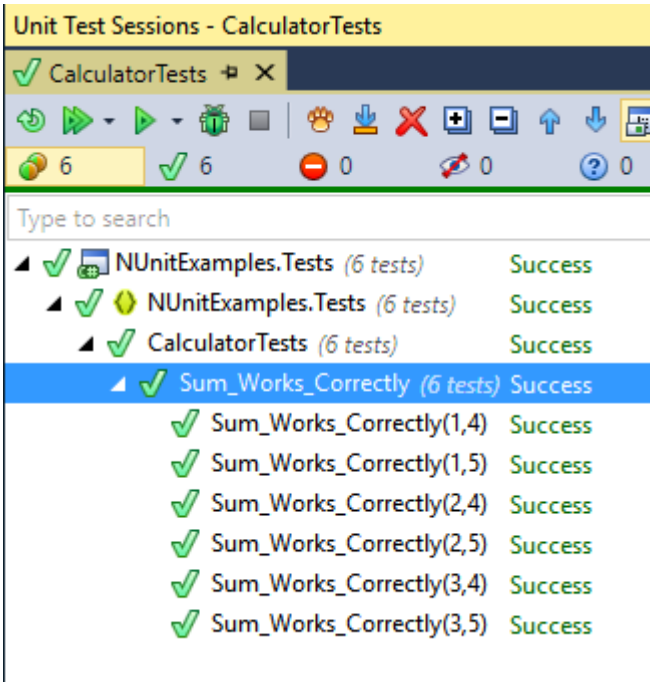
```
 [TearDown]
 public void CleanAfterEveryTest()
 {

 }
```

ValeursAttribut

ValuesAttribute est utilisé pour spécifier *un ensemble de valeurs* pour un paramètre individuel d'une méthode de test avec des paramètres.

```
[Test]
public void Sum_Works_Correctly(
    [Values(1, 2, 3)] int x,
    [Values(4, 5)] int y)
{
    // Arrange
    var calculator = new Calculator();

    // Act
    int result = calculator.Sum(x, y);

    // Assert
    Assert.That(result, Is.EqualTo(x + y));
}
```

Nous pouvons voir ici quels cas de test sont exécutés sur ces valeurs:



Lire Les attributs en ligne: https://riptutorial.com/fr/nunit/topic/6512/les-attributs

Chapitre 5: Tester l'exécution et le cycle de vie

Examples

Exécuter des tests dans un ordre donné

Normalement, vos tests doivent être créés de manière à ce que l'ordre d'exécution ne pose aucun problème. Cependant, il y aura toujours un avantage si vous devez enfreindre cette règle.

Le scénario que j'ai rencontré concernait R.NET, dans lequel, dans un processus donné, vous ne pouvez initialiser qu'un seul moteur R et, une fois éliminé, vous ne pouvez plus le réinitialiser. L'un de mes tests concernait la mise au rebut du moteur et si ce test devait être exécuté avant tout autre test, il échouerait.

Vous trouverez ci-dessous un extrait de code de la façon dont j'ai réussi à faire en sorte que Nunit fonctionne correctement.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using NUnit.Framework;
using RSamples;

public class OrderedTestAttribute : Attribute
{
    public int Order { get; set; }

    public OrderedTestAttribute(int order)
    {
        this.Order = order;
    }
}

public class TestStructure
{
    public Action Test;
}

public class SampleTests
{
    [TearDown]
    public void CleanUpAfterTest()
    {
        REngineExecutionContext.ClearLog();
    }

    [OrderedTest(0)]
    public void Test1(){}

    [OrderedTest(1)]
```

```
    public void Test2(){}

    [OrderedTest(2)]
    public void Test3(){}

    [TestCaseSource(sourceName: "TestSource")]
    public void MainTest(TestStructure test)
    {
        test.Test();
    }

    public static IEnumerable<TestCaseData> TestSource
    {
        get
        {
            var assembly = Assembly.GetExecutingAssembly();
            Dictionary<int, List<MethodInfo>> methods = assembly
                .GetTypes()
                .SelectMany(x => x.GetMethods())
                .Where(y => y.GetCustomAttributes().OfType<OrderedTestAttribute>().Any())
                .GroupBy(z => z.GetCustomAttribute<OrderedTestAttribute>().Order)
                .ToDictionary(gdc => gdc.Key, gdc => gdc.ToList());

            foreach (var order in methods.Keys.OrderBy(x => x))
            {
                foreach (var methodInfo in methods[order])
                {
                    MethodInfo info = methodInfo;
                    yield return new TestCaseData(
                        new TestStructure
                        {
                            Test = () =>
                            {
                                object classInstance =
Activator.CreateInstance(info.DeclaringType, null);
                                info.Invoke(classInstance, null);
                            }
                        }).SetName(methodInfo.Name);
                }
            }
        }
    }
}
```

Lire Tester l'exécution et le cycle de vie en ligne: https://riptutorial.com/fr/nunit/topic/2789/tester-l-execution-et-le-cycle-de-vie

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec nunit	Community, forsvarir, kdtong, Old Fox, Pavel Yermalovich, Thulani Chivandikwa, Woodchipper
2	Assertions Courantes	D.R., kdtong, mark_h, Paul Hicks
3	Écrire une contrainte personnalisée pour le modèle de contrainte	Horia Coman
4	Les attributs	kame, kdtong, Pavel Yermalovich, RJFalconer, Sibeesh Venu
5	Tester l'exécution et le cycle de vie	forsvarir, Thulani Chivandikwa