**Hướng dẫn thực hành kiểm thử phần mềm**

**với Visual Studio 2010 (phần 1)**

# Tạo Unit test trong VS2010

## Unit test:

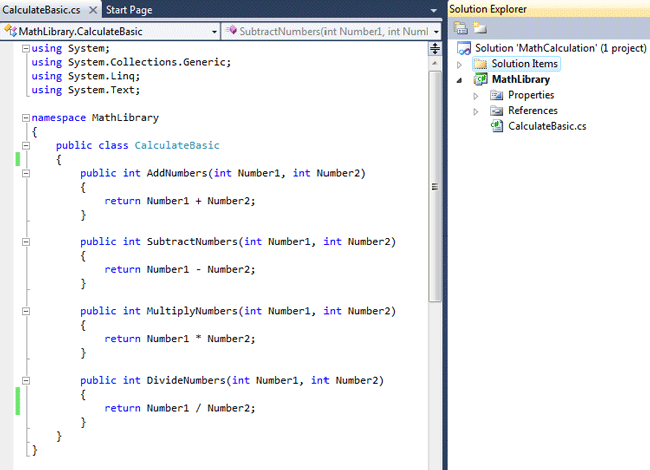
Unit test là kỹ thuật được dùng bởi developer để đảm bảo từng phần đoạn mã hoạt động như mong muốn trước khi tích hợp chúng.

Các bước thực hiện như sau:

<http://www.c-sharpcorner.com/uploadfile/sanks/visual-studio-2010-tools-to-improve-code-quality/>

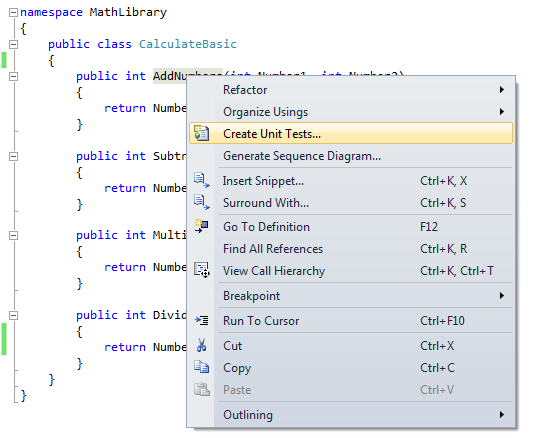
Bước 1: Tạo **class library**

Giả sử class library được đặt tên MathLibrary, trong đó có lớp **CalculateBasic** với các hàm để cộng, trừ, nhân, chia 2 số như hình sau:



Bước 2: Tạo Unit Test

Click phải vào bất kỳ hàm nào, chọn “Create Unit Tests”.



Cửa sổ **Create Unit Test** hiện ra, nhấp chọn vào tất cả các hàm tính, nhấn nút **Setting** để đặt lại tên file, tên class và tên hàm nếu muốn (thường các tên này được hệ thống tự đặt theo tên class của ứng dụng đang test, tên được kết thúc bởi ‘Test’ để phân biệt).

Ở mục General:

*Mark all test results Inconclusive by default*: khi chọn vào mục này, câu lệnh Assert.Inconclusive sẽ được thêm vào tất cả các hàm test phát sinh trong class, điều đó có nghĩa là hàm test đó chưa đầy đủ, chưa được chỉnh sửa cho phù hợp.

*Enable generation warnings*: nếu có lỗi xảy ra trong quá trình phát sinh hàm test thì bộ phát sinh mã sẽ thêm các câu thông báo lỗi dạng comment trong hàm test.

*Globally qualify all types*: giải quyết mâu thuẫn giữa các tên đặt trùng, có thể tự thêm namespace để phân biệt.

*Enable documentation comments*: mục này dùng để phát sinh chú thích cho mỗi hàm. Ví dụ:

/// <summary>

///A test for GetuserDetails

///</summary>

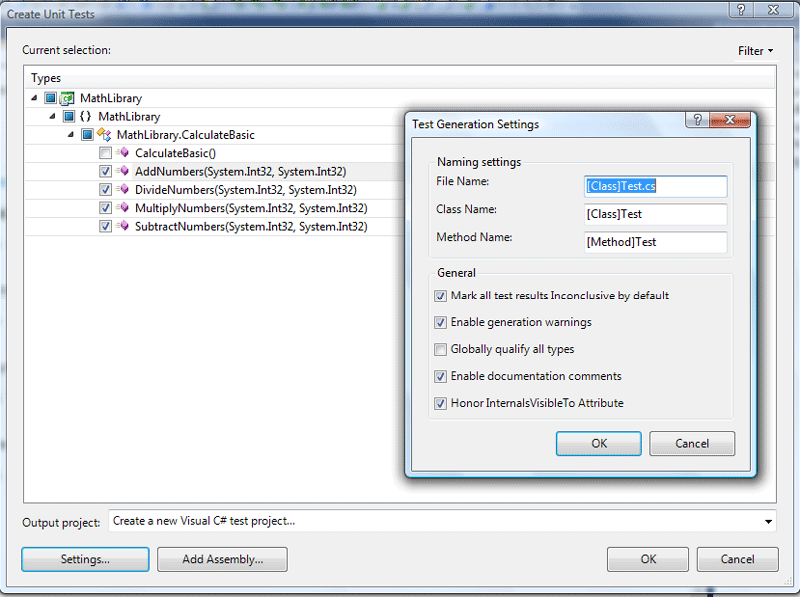
*Honor InternalsVisibleTo Attribute*: thuộc tính này làm cho tất cả các thuộc tính internal trở thành public trong hàm test. Ví dụ:

Có hàm cần test dạng internal như sau:

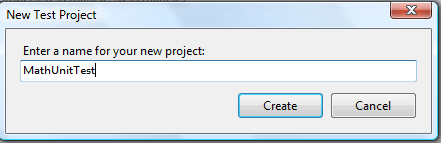
*internal* static bool MethodforInternalExample(string str)

Nếu mục này được check lên thì chương trình sẽ phát sinh hàm test như sau:

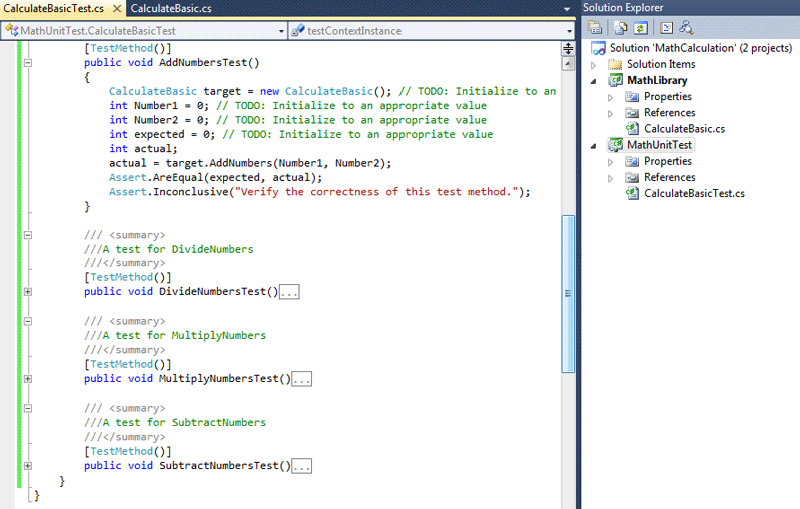
*public* void MethodforInternalExampleTest()



Nhập tên cho project Unit Test.



Cuối cùng, project mới được tạo có tên MathUnitTest với lớp CalculateBasicTest, trong đó có 4 hàm dùng để test cho 4 hàm tính toán tương ứng bên class library. Những hàm phát sinh này chỉ là bộ khung (skeleton). Phải chỉnh sửa các giá trị hay thêm mã cho phù hợp, khi đó cần xóa dòng lệnh Assert.Inconclusive trước khi chạy test.



Câu lệnh **Assert** được dùng để so sánh kết quả của hàm với kết quả mong đợi. Tìm hiểu thêm trong sách trang 85.

Types of asserts

[**Assert**](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assert.aspx)

The assert class has many different overloaded methods for comparing the values. Each method is used for a specifc type of comparison. In your test method, you can call any number of methods of the Assert class, such as Assert.AreEqual().

[**CollectionAssert**](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.collectionassert.aspx)

Use the CollectionAssert class to compare collections of objects, and to verify the state of one or more collections.

[**StringAssert**](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.stringassert.aspx)

Use the StringAssert class to compare strings. This class contains a variety of useful methods such as StringAssert.Contains, StringAssert.Matches, and StringAssert.StartsWith.

[**AssertFailedException**](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assertfailedexception.aspx)

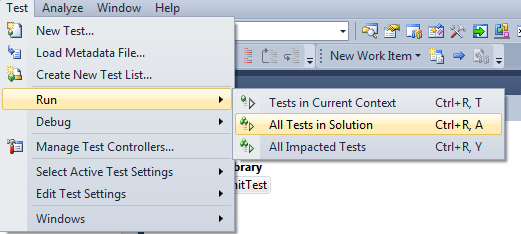
The AssertFailedException exception is thrown whenever a test fails. A test fails if it times out, throws an unexpected exception, or contains an Assert statement that produces a Failed result.

[**AssertInconclusiveException**](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assertinconclusiveexception.aspx)

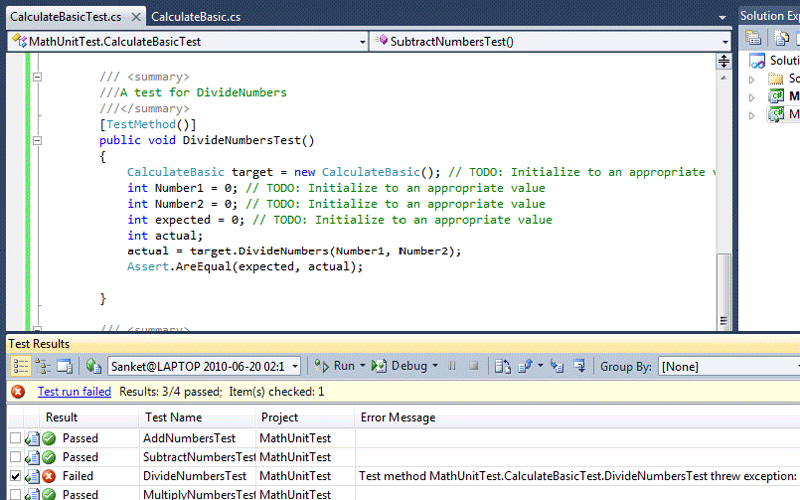
The AssertInconclusiveException is thrown whenever a test produces a result of Inconclusive. Typically, you add an Assert.Inconclusive statement to a test that you are still working on to indicate it is not yet ready to be run.

Bước 3: Chạy Unit Test.

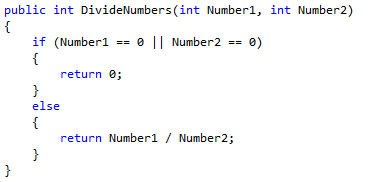
Vào menu Test và chọn Run --> All Test in Solutions. Muốn kiểm từng hàm thì click phải vào hàm muốn kiểm chọn Run Test.



Kết quả test là hàm DivideNumbersTest bị fail vì lỗi chia cho 0.



Sửa lại hàm DivideNumbers trong class library như sau:



Run All Tests lần nữa thì kết quả là tất cả các test đều pass.

Chú ý: Trong mỗi hàm test, các giá trị x, y, expected, actual có thể được sửa để phù hợp với test-case đã thiết kế.

## Tạo Unit Test khi không có source code:

Có thể tạo Unit test từ file dạng .exe hoặc dll. Cách làm như sau:

B1. New Project Test

B2. Add Reference file .dll vào Project Test này (chọn phần assembly và browse đến .dll cần viết unit test)

B3. Thực hiện viết hàm Unit Test (không phát sinh hàm test).

## Kiểm tra Exception

Cách 1:

[TestMethod()]

public void test1()

{

Exception expectedException = null;

try

{

somethingThatShouldThrowAnException();

}

catch (Exception ex)

{

// Catches the exception

expectedException = ex;

}

Assert.IsNotNull(expectedException);

}

Cách 2:

[TestMethod()]

public void test1()

{

try

{

somethingThatShouldThrowAnException();

Assert.Fail(); // raises AssertionException

}

catch (FormatException)

{

// Catches the assertion exception, and the test passes

}

catch (Exception)

{

// Catches the assertion exception, and the test passes

}

}

Cách 3:

[TestMethod()]

public void test1()

{

try

{

somethingThatShouldThrowAnException();

}

catch (FormatException)

{

// Catches the assertion exception, and the test passes

}

catch (Exception)

{

// Catches the assertion exception, and the test passes

}

}

## Thuộc tính ExpectedException:

Thuộc tính này có thể được sử dụng để kiểm tra nếu hàm được test chứa exception. Xem ví dụ sau đây:

Test hàm sau:

public double GetTotalItemPrice(int count)

{

double price = 10.99;

double total;

total = count \* price;

if (total < 0)

{

throw new Exception("The total is less than zero");

}

return total;

}

Sửa hàm test như sau (cho biến count bằng -1):

[TestMethod()]

public void GetTotalItemPriceTest()

{

Class1 target = new Class1();

**int count = -1;**

double expected = 0F;

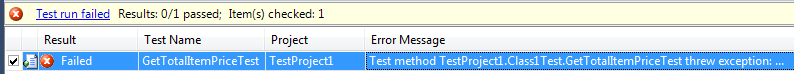
double actual;

actual = target.GetTotalItemPrice(count);

Assert.AreEqual(expected, actual);

}

Khi chạy hàm test này kết quả cho:



Để sửa lỗi này, thêm thuộc tính ExpectedException vào sau [TestMethod()]: [ExpectedException(typeof(Exception))]

**Hướng dẫn thực hành kiểm thử phần mềm**

**với Visual Studio 2010 (phần 2)**

# Tính độ bao phủ mã (Code Coverage) trong Unit test

VS 2013: Chọn menu Test\Analyze Code Coverage

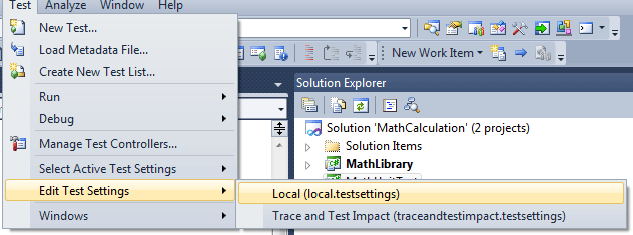
Mở cửa sổ Code Coverage Result để xem kết quả

http://www.c-sharpcorner.com/uploadfile/sanks/visual-studio-2010-tools-to-improve-code-quality/

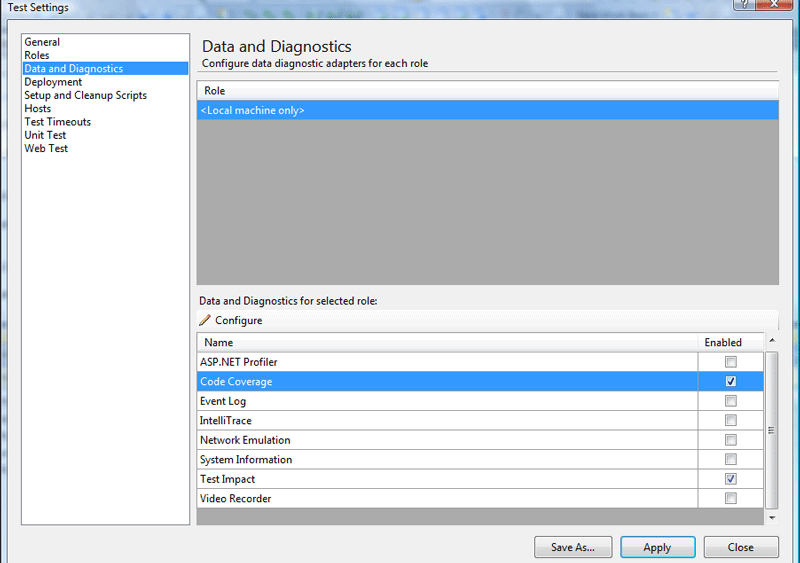
Sau khi thực hiện 3 bước ở mục Unit Test, tiếp tục thực hiện như sau:

Bước 4: Mở cửa sổ Code Coverage

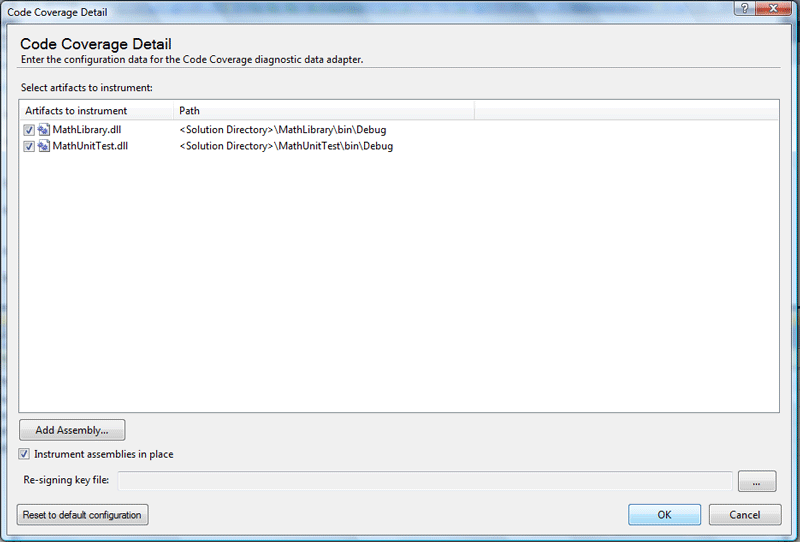
Để mở cửa sổ xem Code Coverage, chọn vào menu Test --> Edit Test Settings --> Local (local.testsettings)



Cửa sổ Test Settings xuất hiện, chọn Data and Diagnostics, check vào Code Coverage và nhấn nút Configure để cấu hình cho Code Coverage.

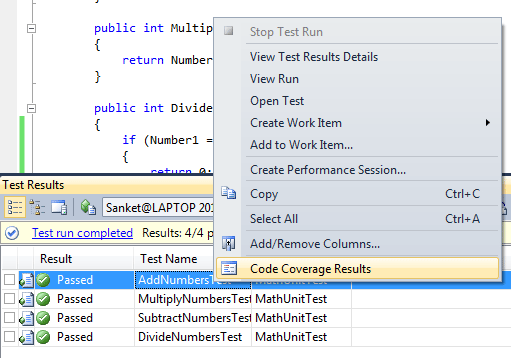


Chọn các assemblies để đưa vào Code Coverage, nhấn OK.



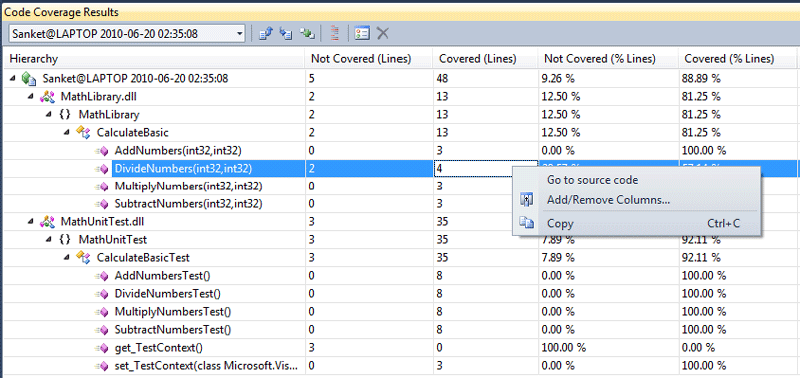
Bước 5: Kiểm tra độ bao phủ mã

Chạy tất cả test lần nữa. Click phải vào kết quả test muốn xem, chọn Code Coverage Results

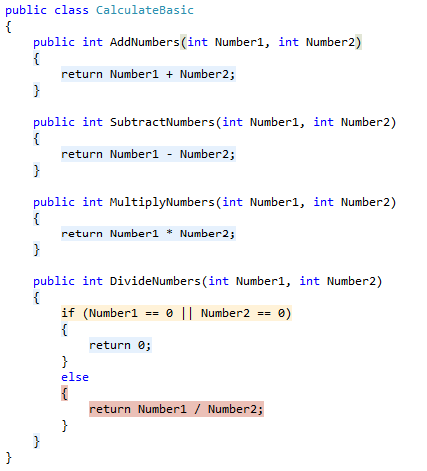


Khi cửa số Code Coverage Results hiện ra, click chọn vào các nút để sổ ra hàm cần xem, kết quả thống kê gồm 4 cột: thống kê số dòng không được thực thi và số dòng được thực thi theo số lượng và theo phần trăm.

Để xem dòng nào chưa được thực thi, có thể click phải vào hàm và chọn Go to source code.



Màu của dòng lệnh được thực thi và chưa được thực thi có sự phân biệt.



**Hướng dẫn thực hành kiểm thử phần mềm**

**với Visual Studio 2010 (phần 3)**

# Thực hiện Data-Driven trong Unit test

<http://www.c-sharpcorner.com/uploadfile/sanks/visual-studio-2010-data-driven-unit-test/>

<http://swautomation.wordpress.com/2011/06/20/visual-studio-2010-ultimate-testing-data-driven-testing/>: Data Driven Testing for Calculator Application

Data Driven Testing là gì?

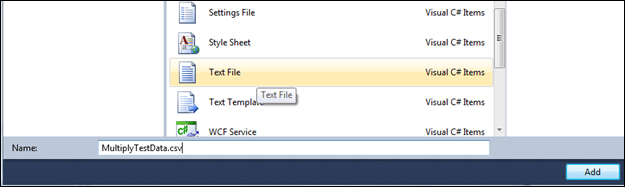
Làm thế nào để chạy các test tương tự nhiều lần với các bộ dữ liệu (test-case) khác nhau mà không cần phải viết nhiều hàm test?

Testing với Data Driven sẽ giúp giải quyết điều này, có nghĩa là khả năng sử dụng cơ sở dữ liệu để chứa các test case đã thiết kế như là nguồn dữ liệu cho Unit test, công việc của tester giờ chỉ là kết nối với cơ sở dữ liệu đó để lấy giá trị đưa vào các biến trong hàm test. Hơn nữa Data Driven còn xử lý cho cả các nguồn dữ liệu khác như file text…

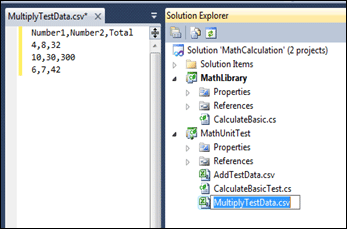
Sau khi thực hiện Unit Test (thực hiện ở mục 1.1), chúng ta thực hiện kết nối file dữ liệu test với các hàm test. Vd/ thực hiện test hàm MultiplyNumbers với dữ liệu test được lấy từ file .csv.

**Bước 1**: Tạo file .csv

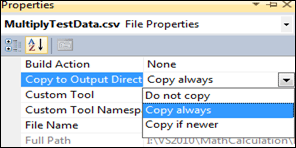
Click phải vào project MathUnitTest và chọn Add -> New Item. Chọn dạng file muốn tạo là Text File và đặt tên MultiplyTestData.csv.



**Bước 2**: Nhập liệu vào file .csv.



**Bước 3**: Thiết lập thuộc tính cho file .csv là Copy always để file được copy vào thư mục bin sau khi build.



**Bước 4**:

* Tạo chuỗi kết nối cho hàm TestMethod():

[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",

"|DataDirectory|\\data.csv", "data#csv", DataAccessMethod.Sequential),

DeploymentItem("data.csv"), TestMethod()]

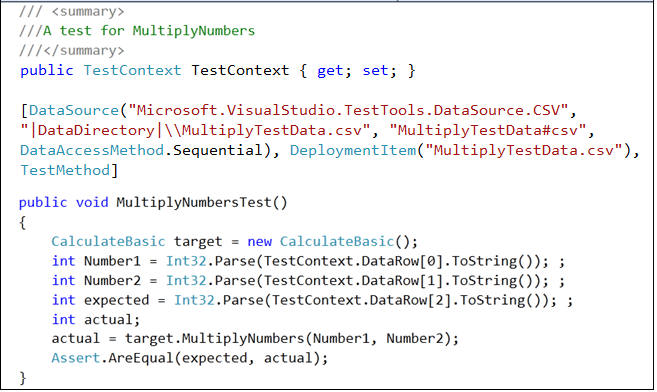
* Ràng buộc dữ liệu:

Tạo TestContext:



Sử dụng thuộc tính TestContext.DataRow[ColumnIndex] (cột đầu tiên có chỉ số là 0) hoặc TestContext.DataRow["NameOfColumn"] để thay đổi giá trị gán cho các biến. Cần Add Reference đến System.Data để sử dụng DataRow.

Ví dụ thay đổi giá trị gán cho Number1, Number2, expected bằng TestContext.DataRow[0], TestContext.DataRow[1], TestContext.DataRow[2]:



**Bước 5**: Thực thi test và ghi nhận kết quả test.

Kết luận: Sử dụng Data-driven khi test Unit giúp cho việc thực hiện test được nhanh chóng, thuận tiện, kiểm tra được nhiều dữ liệu mà không cần phải viết nhiều hàm hoặc sửa nội dung hàm test. Ngoài ra cũng có thể dùng cách này để kiểm tra độ bao phủ mã.

# Coded UI Test

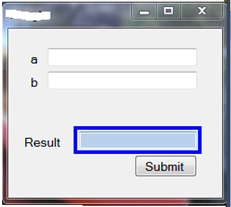
<http://www.youtube.com/watch?v=rZ8Q5EJ2bQM>

<http://msdn.microsoft.com/en-us/library/dd286608.aspx>

## Coded UI Test:

Là cách kiểm tra chức năng của UI bằng cách record và play.

Ví dụ: Cho ứng dụng cộng hai số như sau:



Các bước tạo và làm việc với Coded UI Test:

* Tạo test project.
* Click phải vào test project chọn Add\Coded UI Test… Hộp thoại Generate Code xuất hiện. Chọn record mới hay sử dụng lại đoạn record đã có, OK.
* Click vào nút Start Recording để bắt đầu record: khởi động ứng dụng muốn test rồi thao tác, tắt ứng dụng khi hoàn thành. Bấm nút Generate Code để phát sinh hàm test cho đoạn chương trình vừa record.



* Giả sử đặt tên hàm cho đoạn mã phát sinh là AddTwoNumbers, trong chương trình có sự thay đổi như sau:
  + UIMap.Designer.cs có hàm AddTwoNumbers, đoạn mã trong hàm có thể được sửa đổi hay thêm lệnh:

public void AddTwoNumbers()

{

#region Variable Declarations

WinEdit textInput1Edit =

this.DemoCalculatorWindowWindow.InputNumber2Window.TextInput1Edit;

WinEdit textInput2Edit =

this.DemoCalculatorWindowWindow.TextInput2Window.TextInput2Edit;

WinButton addButton =

this.DemoCalculatorWindowWindow.AddWindow.AddButton;

#endregion

// Launch '%USERPROFILE%\Desktop\SimpleWinformsCalculator.exe'

ApplicationUnderTest demoCalculatorWindowWindow =

ApplicationUnderTest.Launch(

this.AddTwoNumbersParams.DemoCalculatorWindowWindowExePath,

this.AddTwoNumbersParams.DemoCalculatorWindowWindowAlternateExePath);

// Type '3' in 'textInput1' text box

textInput1Edit.Text =

this.AddTwoNumbersParams.TextInput1EditText;

// Type '4' in 'textInput2' text box

textInput2Edit.Text =

this.AddTwoNumbersParams.TextInput2EditText;

// Click 'Add' button

Mouse.Click(addButton, new Point(83, 18));

}

* + CodedUITest1.cs có gọi hàm AddTwoNumbers

[TestMethod]

public void CodedUITestMethod1()

{

// To generate code for this test, select "Generate Code"

// from the shortcut menu and select one of the menu items.

this.UIMap.AddTwoNumbers();

}

* Run test.

Thường khi dùng Coded UI Test có 2 bước hay dùng là khởi động và đóng ứng dụng như ví dụ sau:

[TestMethod]

public void CodedUITestMethod1()

{

this.UIMap.LaunchApp();

this.UIMap.AddTwoNumbers();

this.UIMap.CloseApp();

}

Có thể viết lại như sau (đoạn lệnh được lấy từ file UIMap.Designer.cs):

public void CodedUITestMethod1()

{

**using (ApplicationUnderTest.Launch(**

**this.AddTwoNumbersParams.DemoCalculatorWindowWindowExePath,**

**this.AddTwoNumbersParams.DemoCalculatorWindowWindowAlternateExePath))**

{

this.UIMap.AddTwoNumbers();

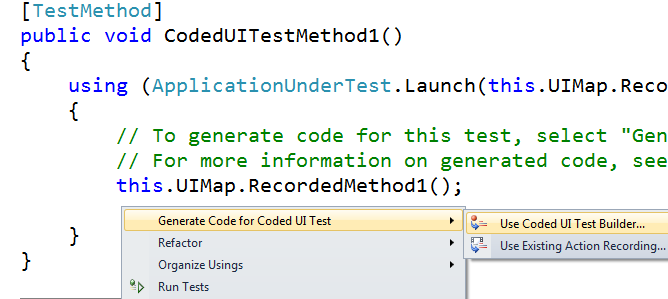
}

}

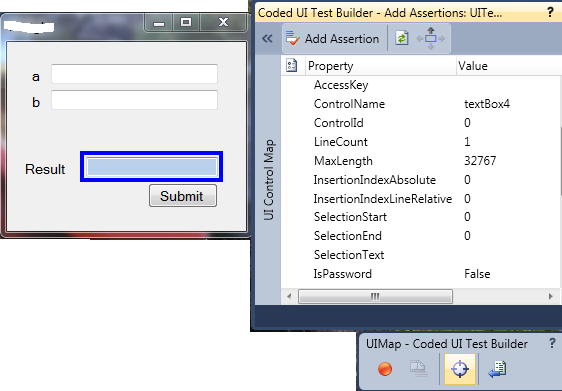
Chú ý: Bấm nút record trước rồi mới kích hoạt ứng dụng, nếu không sau khi record, để run test xem lại thì phải kích hoạt ứng dụng chạy lên trước.

## Phát sinh assertion cho control (kiểm tra các thông điệp trên giao diện):

* Mở file CodedUITest1.cs, đặt con trỏ vào nơi muốn chèn assertion trong hàm CodedUITestMethod1() và chọn Generate Code for Coded UI Test\Use Coded UI Test Builder…

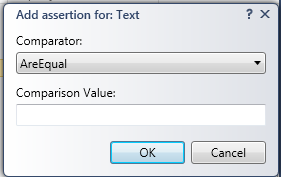


* Coded UI Test Builder xuất hiện, khởi động lại ứng dụng cần test, kéo thả crosshair từ Coded UI Test Builder vào control trên giao diện ứng dụng.
* Sau khi chọn vào 1 control, màn hình Add Assertions xuất hiện liệt kê các thuộc tính ứng với control đó. Chọn thuộc tính muốn kiểm tra và nhấn nút Add Assertion.



**crosshair**

* Chọn loại so sánh và cho giá trị muốn so sánh, OK.



* Click nút Generate code trên Test Builder cho assertion.

## Data Driven Coded UI Test

* Tạo file dữ liệu nguồn chứa các test-case.
* Kết nối file dữ liệu nguồn với Coded UI Test: Test 🡪 Windows 🡪 Test View 🡪 Properties 🡪 Data Connection String 🡪 select csv, XML or database.
* Sửa lại mã trong hàm CodedUITestMethod1 như sau:

**using (…)**

{

**this.UIMap.RecordedMethod1Params.UITextBox1EditText =**

**TestContext.DataRow[0].ToString();**

**this.UIMap.RecordedMethod1Params.UITextBox2EditText =**

**TestContext.DataRow[1].ToString();**

this.UIMap.AddTwoNumbers();

**this.UIMap.AssertMethod1ExpectedValues.UITextBox4EditText =**

**TestContext.DataRow[2].ToString();**

this.UIMap.AssertMethod1();

}

Demo kiểm thử cho chương trình tam giác (TestTriangle).

## Chụp hình form kết quả để kiểm tra:

this.UIMap.AddTwoNumbers();

**Image pic = this.UIMap.UITriangleWindow.CaptureImage();**

**pic.Save(@"D:\file.bmp");**

**TestContext.AddResultFile(@"D:\file.bmp");**

**Hướng dẫn thực hành kiểm thử phần mềm**

**với Visual Studio 2010 (phần 4)**

# Web Performance Testing (Web Test)

## Tạo Web Performance Test

* Tạo Test project
* Chọn menu Test\New Test, chọn **Web Performance Test**. Màn hình **Web Performance Test Recorder** xuất hiện cùng với Internet Explorer.
* Gõ vào tên web site muốn test.
* Nhấn nút **Stop** khi record xong và đóng Internet Explorer.

Khi ‘run test’ với localhost, trước tiên cần phải F5 ứng dụng trước.

A dialog box displays the message **Detecting dynamic parameters**.

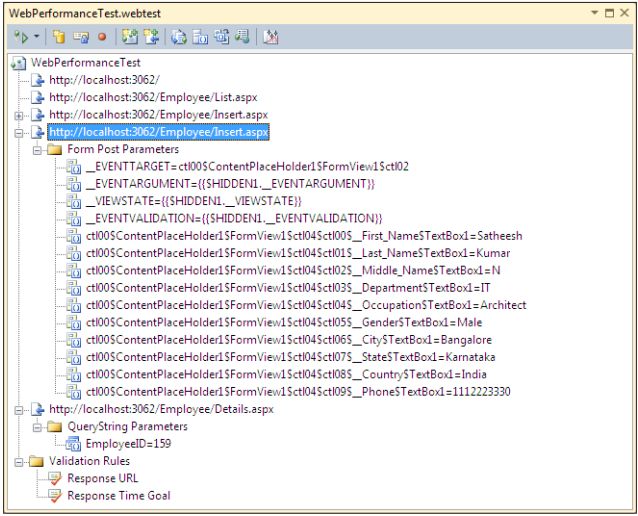
If no dynamic parameters are detected, the dialog box displays the message **Did not detect any dynamic parameters to promote**. The dialog box then closes.

If dynamic parameters are detected, the **Promote Dynamic Parameters to Web Test Parameters** dialog box appears. This dialog box contains a table that lists the dynamic parameter values that were found, if any. You can now promote them to Web test parameters. Each row shows one parameter value that you can choose to promote.

(Optional) In the **Promote Dynamic Parameters to Web Test Parameters** dialog box, select check boxes for the promotions that you want to apply when you run the Web performance test, and click **OK**.

## WebTest editor

Sau khi record xong, màn hình Web Test hiển thị tree view bao gồm tất cả các request thu được khi record. Màn hình này cũng hiển thị các thuộc tính khác nhau của các request và các tham số cho mỗi request. Phần phía trên của WebTest editor là toolbar.

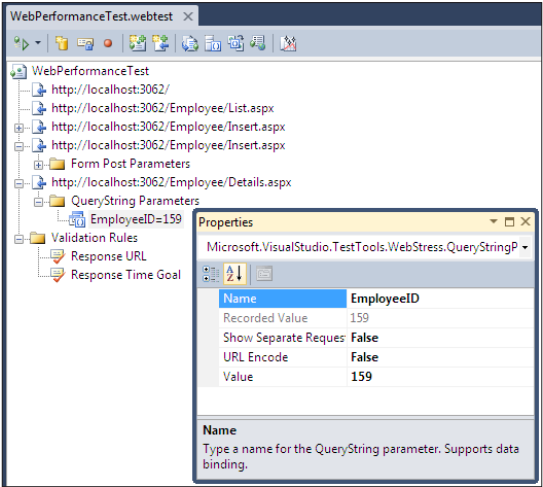


**Form POST parameters**

Chứa các tham số gởi trong request khi request dùng phương thức POST. Giá trị của các tham số này có thể được chỉnh sửa thông qua thuộc tính Value (trong Properties), ban đầu được gán mặc định bằng giá trị trong Recorded Value. Ngoài ra các giá trị này cũng có thể được ràng buộc với nguồn dữ liệu database, XML file hoặc CSV file.

**QueryString parameters**

Chứa các tham số gởi trong request khi request dùng phương thức QueryString. Cách sử dụng tương tự Form POST parameters, ngoại trừ có thêm thuộc tính **Show Separate Request Result**, dùng để nhóm các request dựa trên giá trị của QueryString.

****

**Extraction rules**

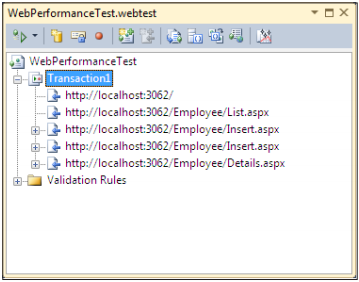
Extraction rules dùng cho việc trích xuất dữ liệu hoặc thông tin từ HTTP response. Các web form trong ứng dụng web thường phụ thuộc lẫn nhau: request dựa trên dữ liệu rút trích được từ response của request trước đó.

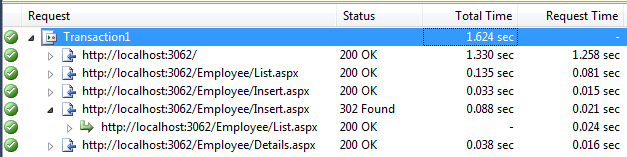
**Validation rules**

Dùng để xác nhận dữ liệu vào hoặc ra.

**Transactions**

Transaction có thể được dùng để nhóm các hoạt động trong web test để dễ theo dõi tổng thời gian của các hoạt động này.





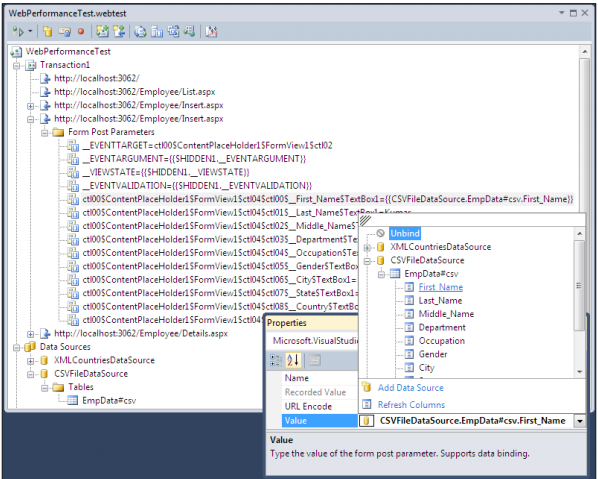
## Tạo data source và ràng buộc dữ liệu

**Tạo data source:** có thể thêm bao nhiêu data source tùy ý

* Nhấn nút Add Data Source trên toolbar Web Test.
* Đặt tên cho data source và chọn loại data source.
* Theo các bước của wizard tùy thuộc vào data source đã chọn.

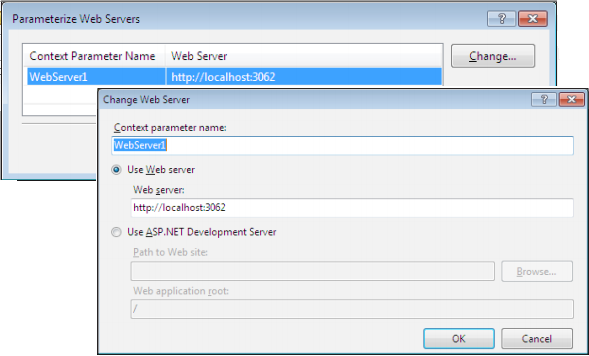
**Ràng buộc dữ liệu trong data source:**

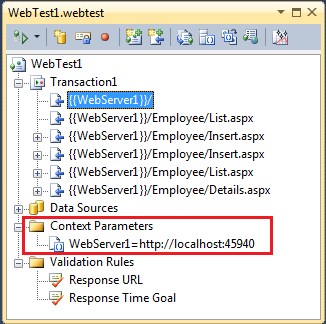
Khi có data source, dữ liệu có thể được ràng buộc cho **Form Post Parameter** hoặc **QueryString Parameter** dưới các request thông qua thuộc tính Value của từng parameter (xem hình).



## Parameterize web server

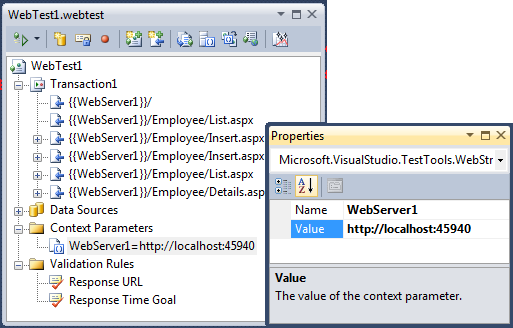
Tất cả các Web Performance Tests được record và thực hiện trên 1 web server. Nếu muốn test trên 1 web server khác thì phải record lại trên web server đó và test lại. Tuy nhiên, VS cung cấp sẵn chức năng Parameterize web server, cho phép chỉnh sửa lại web server.





## Context Parameter

Context Parameter giống như biến toàn cục, với tên biến và giá trị. Có thể thêm nhiều Context Parameter nhưng không được trùng tên. Giá trị của Context Parameter có thể thay đổi thông qua thuộc tính Value.



Giá trị cho Context Parameter có thể được ràng buộc lúc runtime bằng cách sử dụng plug-in.

**Thêm web test plug-in:**

* Tạo một assembly hoặc một class library, phải kế thừa từ Microsoft.VisualStudio.TestTools.WebTesting.WebTestPlugin.
* Thực thi hai hàm PreWebTest() và PostWebTest() trong class.
* Sau khi viết code xong, thêm reference cho Test project.

## Dynamic parameters

# Load Test