TinyUnit: The Simplest Unit Test Framework that Can Possibly Work

Download:

General	For VB6	For VB .NET	For PHP5
General Documentation: This page	Framework (source and DLL)	Framework (source and DLL)	Framework (source=executable)
UML schemes (Poseidon format):	VB6 Scheme	VB .NET Scheme	PHP5 scheme

Why TinyUnit?

After downloading the latest version of *JUnit* and *Nunit*, I was unpleasantly surprised. It had downloaded a huge collection of classes and I did not understand any of it! And I was unit testing for more than two years, so something was definitely wrong. What I expected to be something simple, consisted of more classes then I could possibly imagine for such a simple task. I couldn't even begin trying to understand what all the classes were about. Not even after reading the documentation. What I did understand, was that I would never have a unit test framework set up like that. It appeared to rely heavily on reflection and used singletons (the global variables of object-oriented languages). I had already made my own unit test framework because the existing *VBunit* frameworks all refused to install properly, so I decided that it was time to make my code public. I can't believe that I am the only one having trouble with existing unit test frameworks.
I kept the JUnit terms, but started from scratch, because I was convinced that unit test frameworks did not need the complexity of the existing ones. It is back to basic, back to simplicity, back to understanding. It's what I need as an extreme programmer: the simplest thing that could possibly work.

What's the difference between TinyUnit and the regular unit testing frameworks?

Well, first of all, *TinyUnit* does not use reflection or singletons. This means that your tests aren't "optimised out" of your project before you can even run them, and, more important, that you have control over the order in which the tests are executed. So you can test the independent classes before the classes that depend on them. The advantage, off course, is that a failing independent class will always be the first to fail and therefore be recognisable. The classes that depend on the failing class will likely also fail, so the number of failed tests may be quite large, making it otherwise really difficult to find the real failure.
And, of course, *TinyUnit* is simple. There is a *TestDriver* and there are tests. There's nothing more to it. The *TestSuite* is a group of tests and therefore a test itself.

For what language is TinyUnit written?

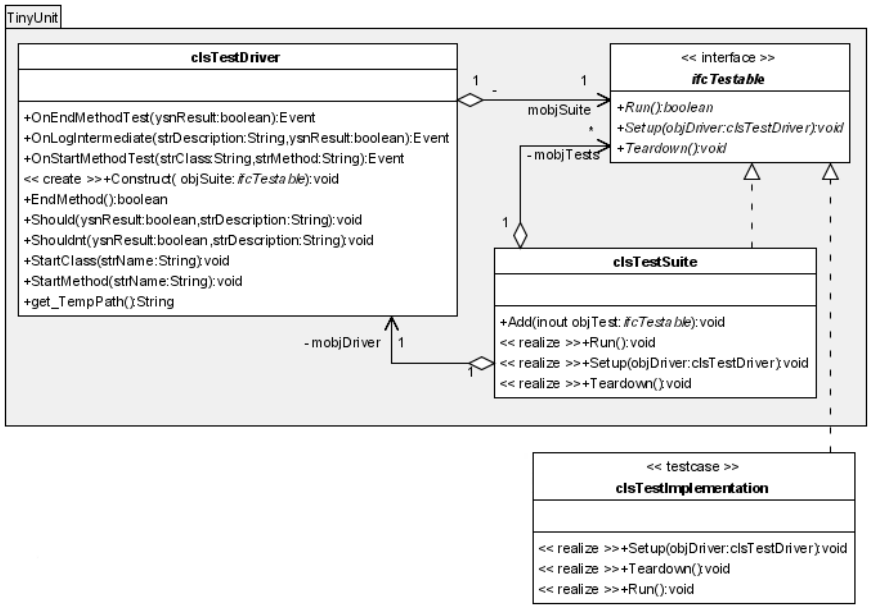
TinyUnit was originally written for Visual Basic 6.0 and recently a version was added for Visual Basic .NET. A version for other object-oriented languages that do not support the Microsoft-style events is easily made, as is shown in the PHP5 package. Java programmers will not have much trouble reading PHP5 source code.

Class overview

TinyUnit consists of only three elements: a testdriver that starts the tests and gathers the results, a testclass, and a testsuite, which is nothing more than a kind of collection. The testclass is not a concrete class: you derive your own testclasses from it.

Visual Basic 6

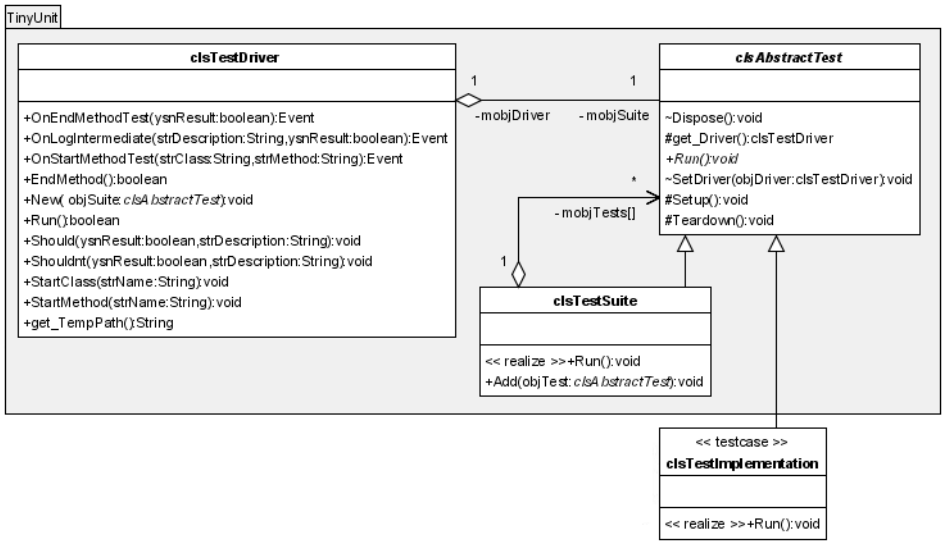
In Visual Basic 6, the testclass can only be an interface (Visual Basic 6 and earlier have no implementation inheritance that you can define yourself), so the UML scheme looks as follows:



Where any method starting with "get_" is a property get routine.

Visual Basic .NET

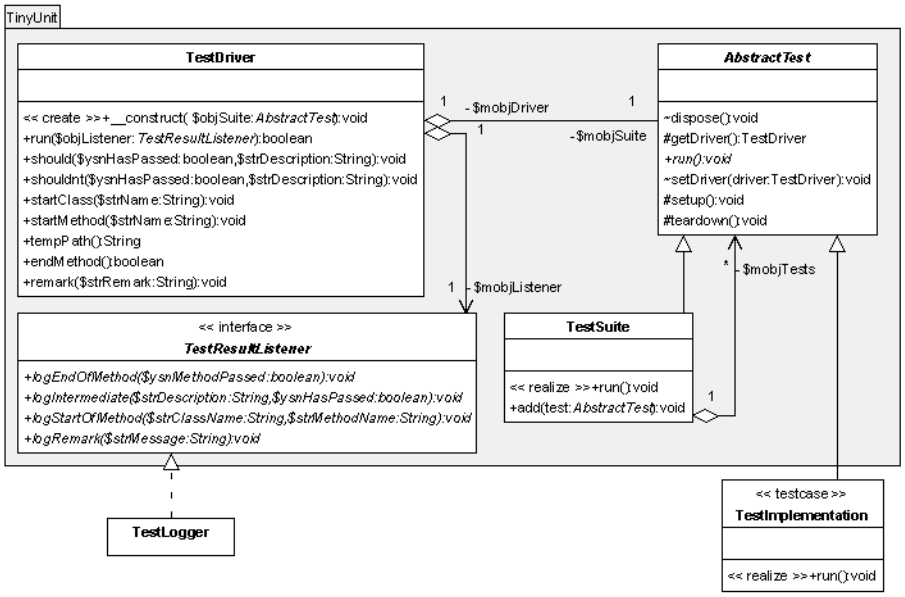
Visual Basic .NET has a better object model that supports abstract classes and "protected" scope. The testclass no longer needs the TestDriver reference to be passed via the interface, and your testclass only needs to override a "Run" method without parameters. The Setup and Teardown methods can also be overridden, but you don't have to if you don't need to. For Visual Basic .NET, the scheme is:



PHP5

In PHP5, Microsoft-style events are not available and java-style events must be used. This adds one listener interface to the model. Your unit tests will probably be run from some sort of logger class that instantiates the TestD*river* and the *TestSuite*. It is called *TestLogger* in this model. The vivibility is shown as the first character in a property or method: "+" denotes a

public method, "-" a private property, "#" means protected and "~" denotes friend. Note that PHP5 does not support friend visibility, so these methods are implemented as public as well. They are shown in the diagram as friend methods to express their meaning.



Some explanation of the framework

The clsTestDriver class

This is the root of the system. It causes the tests to run and gathers all the test results. It also outputs the results to the outside world. The outside world may be a windows form, a web page, the debug console window or just a plain dumb text file. Or any combination of them. That is up to you.

The ifcTestable interface

This is how a test looks from the outside, if you are programming in Visual Basic 6. Visual Basic 6 can only use an interface. Just implement the interface you have a test. You are supposed to store the reference to the *TestDriver* passed through the *Setup* method and report back via this object. Also in the *Setup* method, you can create any other materials you need for your test: a database, a file, objects to be passed as parameters, etc. In the *Teardown* method, release the reference to avoid circular references that cannot be cleared by Visual Basic, and clean up the test material.

The clsAbstractTest class

This is how a test looks from the outside, if you are programming in Visual Basic .NET. Just inherit the abstract class and you have a test. You are supposed to report back to the *TestDriver*, which is available trough the *Driver* method. Override the *Setup* method if you want to set up some test material (files, databases, other objects, etc). The *Teardown* method can be overridden to clean everything up.

The clsTestSuite class

This is nothing more than a collection of test classes. As you will most probably have more than one class to test, you can group all your tests in one or more *TestSuites*.

The test implementations

These are the test classes you write yourself. The tests are, like your code, grouped by classes and methods. This is something to facilitate the location of possible failures; it is not an obligation. However, it is good practice to test each non-private method of a class, because all the non-private methods make up the "contract" of the class. Within the method tests, the actual tests are performed. You are supposed to use the following methods of the *clsTestDriver* class (that was passed to you through the *Setup* method or is available through the *Driver* method):

- *StartClass* does not raise an event, but simply sets the class name.
- *StartMethod* raises an "*OnStartMethodTest*" event with a message containing the class and the method being tested.
- *Should* and *Shouldnt* raise "*OnLogIntermediate*" events that contain the results of the actual testing and a description.
- *EndMethod* raises an "*OnEndMethodTest*" event and returns the state (true=passed, false=failed) of the method test.

The cookbook

Here are the simple steps toward your first test project with TinyUnit.

Define your test classes

VB6 users define test classes that implement the *ifcTestable* interface, VB .NET users define classes that inherit from the *clsAbstractTest* class.

Group them into one or more TestSuites

Define a variable as *clsTestSuite* and store newly created test objects in it.

Create an instance of the TestDriver

Define a "WithEvents" variable to store an instance of the *clsTestDriver* class in. Pass your *TestSuite* to the *TestDriver* in the constructor.

Process the results

Catch the events generated by the *TestDriver* it and display them or log them.