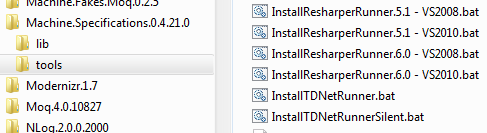
Testing With MSpec

# Setup Instructions

## Setup Test Runners

### ReSharper

Inside the packages folder corresponding to Machine.Specifications, a tools folder is created with a series of assemblies and some batch files. There’s a series of batch files named InstallResharperRunner.X.X. – VS20XX.bat where X’s correspond to the version of ReSharper and of Visual Studio. By executing the corresponding one, it will copy the assemblies to the ReSharper plug-in folder.  


As gloriously simple as this is, I’ve found it doesn’t seem to copy the files to the correct locations. Close Visual Studio and re-open it. Then go to ReSharper => Options => Plugins to see if the MSpec plugin is listed. If it is not listed, click on the “Show developer information” link to see what happened (just because it’s cool!).

Then copy the Machine.Specifications.ReSharperRunner.X.X.dll for your version of ReSharper to

C:\Program Files (x86)\JetBrains\ReSharper\v6.0\Bin\Plugins

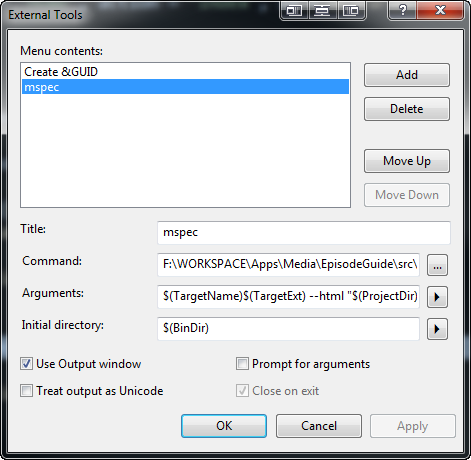
Now the plugin should be listed. You need to have this working for the steps that follow, but unfortunately I can’t run my unit tests with ReSharper! Let me know if you find the solution and I will update this document!

### TestDriven.Net

In the Machine.Specifications tools folder is also a batch file named InstallTDNetRunner.bat. Executing this file will install the TestDriven.Net runner.

### MSpec

Add the mspec test runner to Visual Studio to run tests and produce HTML report. Go to Tools/External Tools and add the following External Tool as you see below:



Note that the command box points to the ConsoleRunner in the mspec tools folder.

packages\Machine.Specifications.0.4.21.0\tools\mspec-clr4.exe

If you want an HTML report to be output to your project – you can do it by adding this as your arguments, which will tell MSpec to output a file called “Report.html”.

$(TargetName)$(TargetExt) --html "$(ProjectDir)\Report.html"

Add this External Tool to a toolbar (if in doubt ask Rob)

<http://blog.wekeroad.com/2007/01/13/subsonic-console-groovy-vs-shortcuts>

## Adapt Visual Studio Environment

### Import resharper live templates:

Inside the packages folder corresponding to developwithpassion.specifications.moq there is a *templates* folder with the following templates:

* file\_templates\_developwithpassion\_specifications\_moq.xml
* live\_templates\_developwithpassion\_specifications.xml
* live\_templates\_machine\_specifications.xml

Open ReSharper live templates menu (ReSharper: Live Templates)

On the *file templates* tab select the User Templates node and then click the import icon and navigate to file\_templates\_developwithpassion\_specifications\_moq.xml and click open.

Next, select and drag the ObservationsWithContract, ObservationsWithoutContract, and StaticObservations templates over to the right side window and drop them on the C# projects section. Now you will be able to create these when using resharpers alt + insert shortcut to add new files.

On the *live templates* tab, import both live template files into User Templates: C#

live\_templates\_developwithpassion\_specifications.xml:

This provides the following ReSharper shortcuts:

deps – for providing mocks that are automatically dependency injected to your sut

fake – for providing all other mocks

live\_templates\_machine\_specifications.xml:

This provides the following ReSharper shortcuts:

bddb – because block

bddc – establish block

tc – it block

sa – subject class

bddcu – cleanup

### Preventing ReSharper from marking specifications as unused

By default, ReSharper will think that specification classes (those marked with the [Subject] attribute), and their internals are unused. To change this behavior in Visual Studio:

1. Open the ReSharper Options (ReSharper -> Options...)
2. Select "Code Annotations"
3. Ensure that the namespace "Machine.Specifications.Annotations" is checked
4. Click "OK"

### Change ReSharper formatting options

Make the following changes to your ReSharper formatting options: (ReSharper: Options)

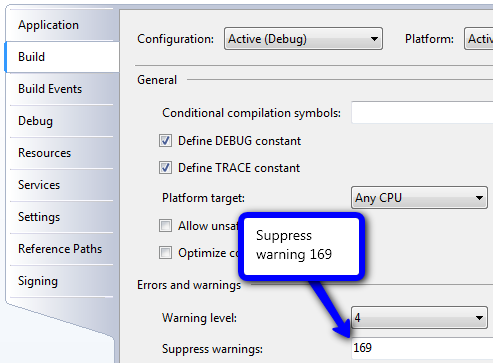
Languages: C#: Formatting Style: Braces Layout

* Anonymous method declaration: At next line (BSD style)
* Array and object initializer: At next line (BSD style)

Languages: C#: Formatting Style: Other

* Modifiers: Use explicit private modifier: Uncheck
* Align Multiline Constructs: Expression: Uncheck
* Align Multiline Constructs: Array, object and collection initializer: Uncheck
* Other: Indent anonymous method body: Uncheck
* Other: Indent array, object and collection initializers: Uncheck

Go to your project properties>Build and suppress warning 169



For more on this see:

<http://codebetter.com/aaronjensen/2008/10/19/getting-resharper-and-vs-to-play-nice-with-mspec/>

Fix ReSharper warnings  
<http://www.aspiringcraftsman.com/2010/02/11/resharper-naming-style-for-machine-specifications/>

### Add macro to help with underscores

Add Visual Studio 2010 macro to allow typing “When writing a unit test” and have it turned into When\_writing\_a\_unit\_test. Instructions can be found in this video and the macro is below.  
<http://www.pluralsight-training.net/community/blogs/starr/archive/2010/05/17/boxcar-case-macro-in-visual-studio-2010-video.aspx>

Imports EnvDTE80

Imports EnvDTE90

Imports EnvDTE90a

Imports EnvDTE100

Imports System.Diagnostics

Imports System.Windows.Forms

Imports System.Text.RegularExpressions

Public Module BddMethods

Sub BoxcarTheCurrentLineOfCode()

Dim description As String

Dim regEx As Regex = New Regex("""([^""]+)""")

If DTE.ActiveDocument Is Nothing Then Return

Dim currentLine As TextSelection = GetCurrentLine()

If (currentLine Is Nothing Or currentLine.Text.Length() < 1) Then Return

Dim stringMatch As Match = regEx.Match(currentLine.Text)

For Each stringMatch In regEx.Matches(currentLine.Text)

currentLine.Text = currentLine.Text.Replace(stringMatch.Value, BoxcarAString(stringMatch.Value))

Next

currentLine.EndOfLine()

End Sub

Private Function BoxcarAString(ByVal textSegmentToChange As String) As String

Dim segmentAfterTheChange As String = textSegmentToChange.Trim()

segmentAfterTheChange = segmentAfterTheChange.Replace("""", "")

segmentAfterTheChange = segmentAfterTheChange.Replace(" ", "\_")

Return segmentAfterTheChange

End Function

Private Function GetCurrentLine() As TextSelection

Dim currentLine As TextSelection = DTE.ActiveDocument.Selection()

currentLine.StartOfLine(vsStartOfLineOptions.vsStartOfLineOptionsFirstColumn)

currentLine.EndOfLine(True)

Return currentLine

End Function

End Module

You can assign this macro to a shortcut key or to a toolbar

<http://blogs.msdn.com/b/kirillosenkov/archive/2008/05/02/how-to-map-a-visual-studio-macro-to-a-keyboard-shortcut.aspx>

# MSpec Unit Test Project

* Created a solution folder for Tests
* Created new project for Unit tests with the name Unit, to follow the convention of the application projects (Web, Infrastructure, etc.)
* Set the namespace to Tests.Unit.Parliament.Events.Committees
* Created folders for each application project – Web, Infrastructure,
* This convention enables the test projects to keep the same namespaces as the application projects, with the prefix of Tests.Unit (and Tests.Acceptance, Test.Integration, etc.). For example  
  Parliament.Events.Committees.Web.Controllers  
  Tests.Unit.Parliament.Events.Committees.Web.Controllers
* Used NuGet to install developwithpassion.specifications.moq package, which also downloaded
  + Moq 4
  + Machine Specifications 0.4
  + Machine Fakes 0.2
  + Machine Fakes Moq

# MSpec Usage

## General Conventions

* Use underscores in names (these are then replaced with spaces for reporting).
* Multiple lines in context
* Generally single line in When
* Multiple Then statements
  + Use the word Should

## Method Comparison

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Establish initial context** | **Interact with System Under Test** | **Check the results** |
| User story structure | Given | When | Then |
| Arrange Act Assert | Arrange | Act | Assert |
| Context Specification  (Machine Specifications) | Establish (context) | Because (of) | It Should |

**Establish** is used to setup the actual scenario. The  
**Because** is where the actual action takes place. Finally the  
**It** sections are where the assertions happen, where we verify the behaviour.

## Tips (from MSpec home page)

### Make your Its/Becauses single-line statements

MSpec is designed to reduce noise in tests. You should generally only have one line in your It and Because statements. As such, you should probably leave out the { and }. Context/Specification testing, while a rethinking of "classic" TDD, still abides by the rules of Arrange-Act-Assert. As such, it is preferable to keep the latter two as succinct as possible.

#### Because

If you're consistently finding that you need to have multiple lines in your Because statements, that may be a code smell that your Context is "too chunky". You want to be able to verify and "nail down" that the behaviour that you're verifying with your It statements is because of a single action. Of course, most non-trivial contexts require multiple lines of setup to get into the preferred state that you're verifying, but you want to be able to say that there is a particular, final action that makes your specifications pass.

For example:

public class because\_example\_goes\_here : ExampleSpecs {

}

#### It

Simultaneously, your It statements should be single-liners and reflect a single assertion. If you're stuffing multiple assertions into a single It, considering the wording of that It and how you may be able to break it up into two or more specifications, each containing a single statement.

public class it\_example\_goes\_here : ExampleSpecs {

}

### Testing for exceptions

When testing for exceptions it is recommended that you use the Catch class in your Because statement then validate the exception in subsequent It statements. This ensures that no validation of the Exception occurs in the Because statement. It is also recommended to include an It statement that explicitly determines the type of Exception if that is important to you. Doing so will improve the readability of your specifications clarifying how the system is intended to behave.

public class when\_the\_user\_credentials\_cannot\_be\_verified : ExampleSpecs {

static Exception Exception;

Because of = () =>

Exception = Catch.Exception(() => SecurityService.Authenticate("user", "pass") );

It should\_fail = () =>

Exception.ShouldBeOfType<AuthenticationFailedException>();

}

# Links

## MSpec

Official github setup instructions

<https://github.com/machine/machine.specifications>

MSpec forums:

<http://groups.google.com/group/machine_users>

Latest status of project

<http://codebetter.com/aaronjensen/2011/03/10/machine-specifications-mspec-0-4/>

Transitioning from xunit to machine.specifications

<http://www.beingnew.net/2010/08/moving-from-nunit-to.html>

MSpec Behaviours:

<http://jagregory.com/writings/behaviours-in-mspec/>

## Background on MSpec Extensions

|  |  |
| --- | --- |
| Machine Fakes | <http://www.adventuretechgroup.com/2011/03/making-machine-specifications-less-machiney/> |
| MSpec.Mvc | <http://jamesbroo.me/introducing-machinespecificationsmvc/> |
| develop with passion specifications | <http://paceyourself.net/2011/06/05/getting-started-with-develop-with-passion-specifications-for-unit-testing/>  <http://blog.developwithpassion.com/2011/03/introducing-developwithpassion-specifications>  <http://blog.developwithpassion.com/2011/04/creating-the-sut-explicitly-in-developwithpassion-specifications>  <http://blog.developwithpassion.com/2011/05/new-feature-for-developwithpassion-specifications-nested-contexts>  *The Examples project has a lot of educational examples*  <https://github.com/developwithpassion/developwithpassion.specifications/tree/master/source/developwithpassion.specifications.examples> |

## MSpec Integration

|  |  |
| --- | --- |
| ReSharper | <http://lostechies.com/seanbiefeld/2009/08/26/step-by-step-to-using-machine-specifications-with-resharper/>  *CODING STANDARDS USING RESHARPER INSTEAD OF STYLE COP*  <http://gojisoft.com/blog/2010/05/10/coding-standards-using-resharper/>  Share ReSharper settings for project in source control  <http://rsm.codeplex.com/> |
| SpecFlow | <http://ntcoding.blogspot.com/2011/01/total-bdd-in-aspnet-mvc_16.html> |
| TeamCity | <http://blogs.jetbrains.com/dotnet/2010/12/coverage-with-dotcover-teamcity-mstest-nunit-or-mspec/> |

## Things to make you go hmmmm

Test Driven Requirements with MSpec

<http://www.skimedic.com/blog/post/2010/01/20/Test-Driven-Requirements-with-MSpec.aspx>

Use the Given When Then syntax with MSpec rather than Establish Because It

<http://www.adomokos.com/2010/09/making-your-tests-more-readable-with.html>

## Test Data with the Builder Pattern

|  |  |
| --- | --- |
| NBuilder | <http://nbuilder.org/> |
| QuickGenerate | <http://davybrion.com/blog/2011/03/random-test-data-anyone/>  <https://github.com/kilfour/QuickGenerate> |
| AutoFixture | <http://blog.ploeh.dk/2010/11/22/IntegratingAutoFixtureWithObjectHydrator.aspx> |

## TDD Katas

To practice TDD

|  |  |
| --- | --- |
| Coding Kata | <http://codingkata.org/> |
| Project Euler | <http://projecteuler.net/> |
| MVC Controllers | <http://codingsolutions.blogspot.com/2010/03/tdd-kata-for-aspnet-mvc-controllers.html> |
| String calculator | <http://osherove.com/tdd-kata-1/> |