

# Introduction

I am going to use this forum for sharing my own best practices in build engineering that I believe are essential for you to be an effective build engineer. This month we will take a look at Microsoft .net and MSBuild. In the future we will cover articles on Ant, Maven and Make. I also provide a link to a related article that I published on CM Crossroads.

## The Challenge

{quote}There have been many times when I had to automate a build using the robust Microsoft .net framework.{/quote} Understanding the .net framework is not easy and I have had a difficult time staying on top of this technology which, to me, truly personifies a moving target. Now to be fair, I spend much more of my time working on high end Java applications running on AIX, Solaris and Linux so I may not always stay ahead of the curve with regard to Microsoft technologies. But then again, it has been my experience that many experienced Microsoft gurus have a hard time staying on top of every aspect of this robust platform too. This has important implications for the build engineer.

## Hiding Behind the IDE

Most C#/.net programmers stay within the comfort of their Visual Studio IDE which has many features and is certainly a very productive environment. The problem is that they may not even be aware of all of their compile and runtime dependencies. That is a problem. Many regulatory bodies insist that there be a separation of duties which means that the build engineer must independently build, package and deploy the application. Implicit in this requirement is that the code should be baselined (e.g. tagged, labeled) and safeguarded in a secure and reliable version control system (VCS). Relying upon the Visual Studio IDE for building does not work in this case because someone could change any number of settings making it impossible for you to build (and support) an older release (obviously needed for addressing bug fixes).

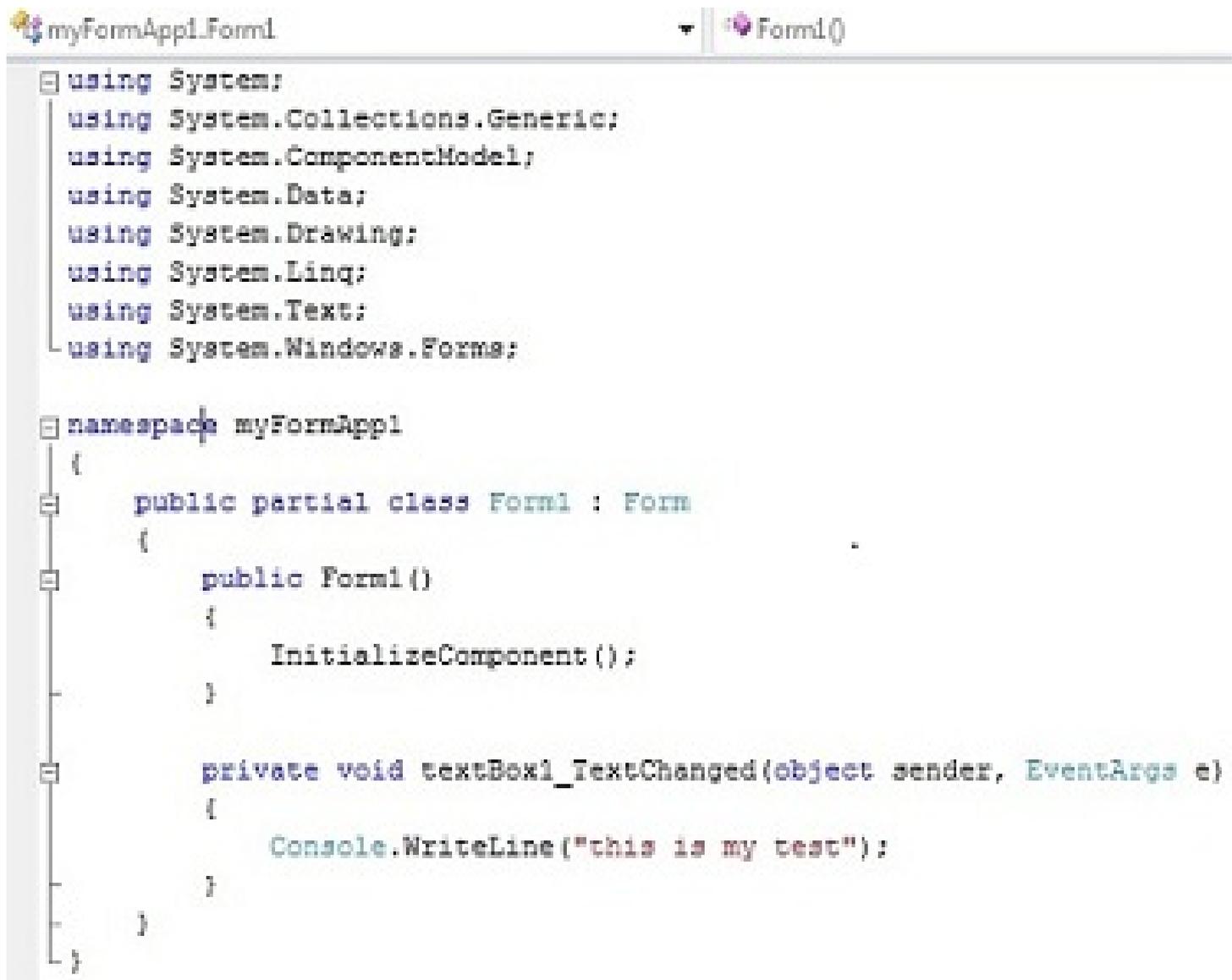
## Embrace Your Command Line

This requires that you build the application from the command line and that the build scripts and procedures be documented and also placed in the version control system labeled with the same tag or label as the code.

I have had many developers look at me with a blank expression when I mentioned that I needed to have a command line procedure. My super smart colleagues spent so much time in the Visual Studio IDE that they did not even know that there was a similar build procedure that could be executed from the command prompt.

### Small Example

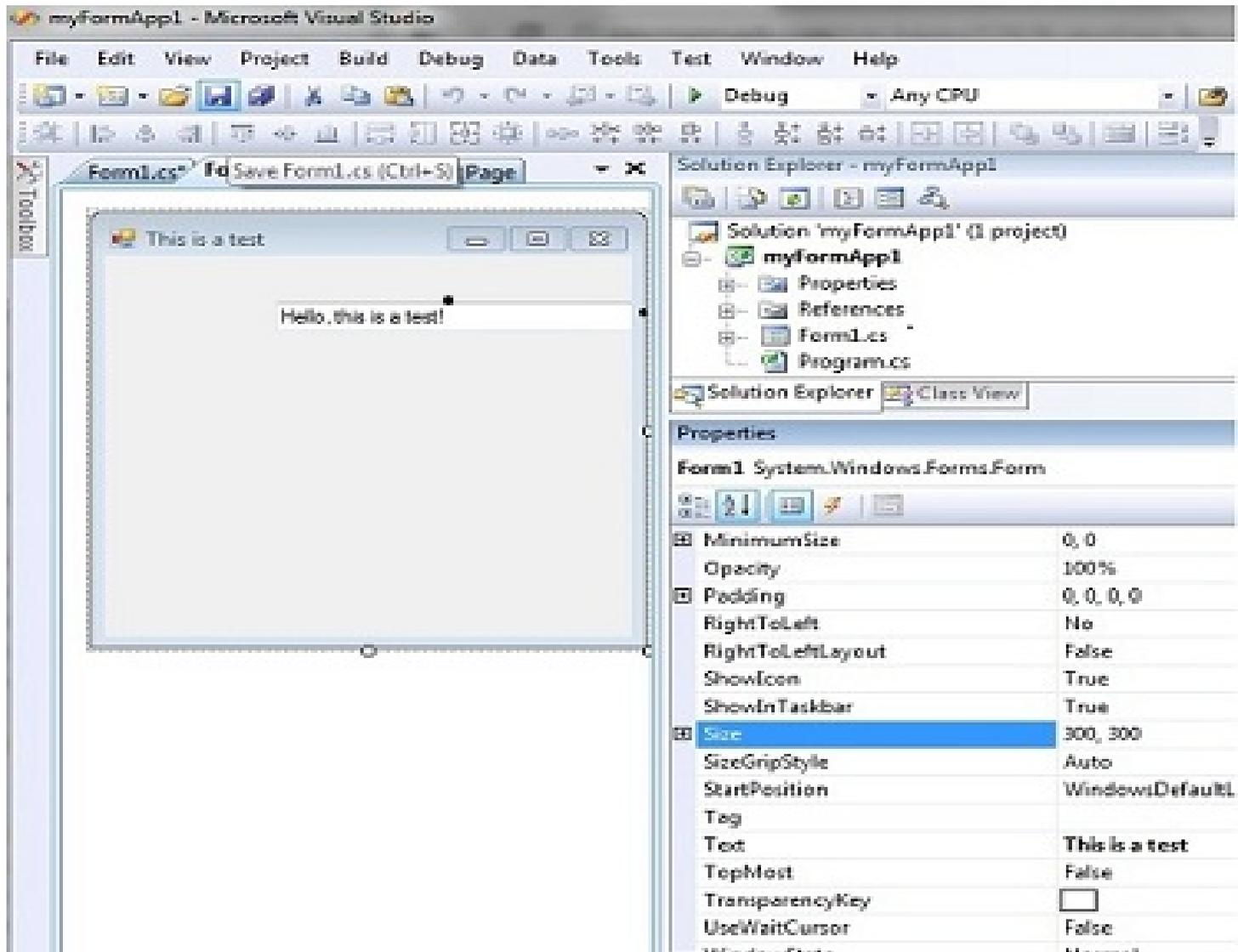
For this article I wrote a tiny program using Microsoft Visual Studio 2008. Here is the code and the form.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace myFormApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            Console.WriteLine("this is my test");
        }
    }
}
```



If you look closely you will see that the IDE support options for a batch build. The resulting code may be found in a subdirectory used by your IDE (hint - do a "save as" and see where you saved the project and code).

You should see a Debug directory and a Release directory with the binary executable that was build.

C:\_sandboxVSProjectsmyFormApp1myFormApp1binRelease

01/15/2012 06:46 PM            7,680 myFormApp1.exe  
01/15/2012 06:46 PM            19,968 myFormApp1.pdb

To test this use the IDE to (batch) "clean" - which refers to removing the binaries that were built (similar to a clean directive in Ant or Make).

Then you can use MSBuild to build the project from the command line using the .sln file which contains all of the information needed for the project. (Note that you may need to update your path to find the MSBuild.exe - mine was in C:WindowsMicrosoft.NETFrameworkv3.5)

C:\_sandboxVSProjectsmyFormApp1>**MSBuild myFormApp1.sln**

Microsoft (R) Build Engine Version 3.5.30729.5420

[Microsoft .NET Framework, Version 2.0.50727.5448]

Copyright (C) Microsoft Corporation 2007. All rights reserved.

Build started 1/15/2012 8:03:30 PM.

Project "C:\_sandboxVSProjectsmyFormApp1myFormApp1.sln" on node 0 (default targets).

Building solution configuration "Debug|Any CPU".

Project "C:\_sandboxVSProjectsmyFormApp1myFormApp1.sln" (1) is building

"C:\_sandboxVSProjectsmyFormApp1myFormApp1myFormApp1.csproj"

(2) on node 0 (default targets).

Processing 0 EDMX files.

Finished processing 0 EDMX files.

CoreResGen:

Processing resource file "Form1.resx" into "objDebugmyFormApp1.Form1.resources".

Processing resource file "PropertiesResources.resx" into

"objDebugmyFormApp1.Properties.Resources.resources".

**CopyFilesToOutputDirectory:**

**Copying file from "objDebugmyFormApp1.exe" to "binDebugmyFormApp1.exe".**

**myFormApp1 ->**

**C:\_sandboxVSProjectsmyFormApp1myFormApp1binDebugmyFormApp1.exe**

**Copying file from "objDebugmyFormApp1.pdb" to "binDebugmyFormApp1.pdb".**

Done Building Project "C:\_sandboxVSProjectsmyFormApp1myFormApp1myFormApp1.csproj" (default targets).

Done Building Project "C:\_sandboxVSProjectsmyFormApp1myFormApp1.sln" (default targets).

Build succeeded.

0 Warning(s)

0 Error(s)

Time Elapsed 00:00:01.14

### Conclusion

Now you can place the entire solution into a version control system (VCS) and test the build from the command line using MSBuild (or Nant). Remember to create your sandbox by pulling the baselined release directly from the VCS using the locked tag or version label. Please drop me a line and tell me about your own best practices in build engineering and any topic related to configuration management and application lifecycle management (ALM)!

Here is another [article](#) that I wrote on this topic some time ago for CM Crossroads.

