Visual Studio 2012 Introduction

Overview

I ran across a useful free eBook recently, *Getting Started with the new Visual Studio 2012*.

The announcement is at <http://p2p.wrox.com/content/blogs/jminatel/free-wrox-ebook-getting-started-new-visual-studio-2012>.

The eBook itself is at <http://p2p.wrox.com/content/sites/default/files/users/21/file/Free%20Wrox%20ebook%20Getting%20Started%20with%20the%20new%20Visual%20Studio%202012.zip>

One warning, though. As the announcement page says, this 70+ page book is the first 5 chapters of the much larger (over 1,000 pages) *Professional Visual Studio* book. However the table of contents in the abbreviated book contains the TOC for the full Pro book and is thus a bit misleading.

A couple of other considerations…

* I haven’t really read this book since I pretty well know it already. So some of my comments may be a bit off.
* I’m sort of assuming that the reader of this document is using the Visual Studio Express edition. But since this is the first 5 chapters of a Pro VS book, some aspects may not be the same (or even available) in the Express edition.
* The book uses the WPF (Windows Presentation Foundation) classes for its examples. WPF is the successor (more or less) to the Windows Forms classes. As such, it is richer than Windows Forms, but also somewhat more complex. So sure, follow the book and do what it says. But once you’re up to speed, you might find the Windows Forms classes (project type *Windows Forms Application*) easier to use (mostly).

Super Quick Overview of Projects and Solutions

Note: The following is a simplified description of projects and solutions, and is meant to give you a feel for what these are, rather than a reference manual of all the details. So with that said…

A Visual Studio *project* represents an executable file, either an exe or a dll.

A Visual Studio *solution* is a set of one or more projects.

Your initial programs will presumably consist of a single exe file, so you’ll have one solution and one project.

In a more advanced scenario, you might be developing an exe and two dlls. You’d have one solution, and three projects (one for the exe, and two for the dlls).

WPF vs. Windows Forms

On the assumption that a programmer new to Visual Studio and C# will, at least initially, develop his applications using Windows Forms, I’ve added this section to help ease the eventual transition from WPF (as described in the Getting Started pdf file) to Windows Forms.

For our purposes, there are two main differences between the two.

* The classes you use are different.
	+ A detailed (or even cursory) description of the classes is beyond the scope of this introductory document) and won’t be described further.
* The GUI builder is, in many respects, very different.

In both Windows Forms and WPF you design your GUI by dragging a control (Button, TextBox, etc) from a toolbox onto a design surface.

In Windows Forms, VS will write out the properties of each control to a source file that is compiled as part of your project. For example, suppose you drag a Button called *Button1* onto the design surface at (x, y) location (50, 100), of width 75 and have it display *Click Me* on the button. VS will generate code similar to the following.

Button Button1 = new Button();

Button1.Location = new Point(50, 100);

Button1.Width = 75;

Button1.Text = “Click Me”;

Whereas WPF will save this information into a text file in the XAML (pronounced *zammel*) format. XAML is based on XML. Our button definition might look like…

<Button Content="Click Me" HorizontalAlignment="Left" Margin="100,50,0,0"

VerticalAlignment="Top" Width="75" RenderTransformOrigin="0.471,-0.332"/>

So the fundamental difference is that Windows Forms compiles your GUI from source code, while WPF uses a data-driven approach.

The good news is that, in both cases, you can mostly ignore exactly how VS saves your GUI definition, and just drag and drop controls from the toolbox.

References

In C and C++, there are .h text files that contain (among other things), prototypes that declare function names, parameters and return types. However, this has the disadvantage that the prototypes are in one file (normally not available at runtime), while the executable (perhaps an older version) might not match the information in the header file.

 .NET approaches this differently. Every executable file (compiled from a .NET language such as C#) contains metadata that describes, in detail, all its classes, methods (i.e. subroutines), parameters (including data type information) and so on.