Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers
Developers

Developers Developers Developers Developers

Print copy available for order. Details at **http://devshaped.com/book**.

Created by Derek Hatchard and Dirk Primbs
Edited by Derek Hatchard
Comics by Sean Forney

Version 1.0

Copyright 2009 Microsoft Corporation

Released under Creative Commons Attribution-Noncommercial-No Derivative Works 3.0

a

{ **You Shape** } It

creation

microsoft.com/youshapeit/msdn

(cc) BY-NC-ND

Contents

Preface

by Dirk Primbs

My first computer was a Sinclair ZX 81. It is the type of computer that makes IT people older than 30 sigh and start talking about their own first computers. C64s, ATARI STs, Amiga… and given the right crowd this discussion surprisingly often leads to the point where we talk about how and - more importantly - with which language we came to programming.

In my case it was the BASIC ROM in the Sinclairs ROM that taught me fundamental programming principles. My parents quickly discovered that their son was in love with computers and to give that a more sincere touch they decided to buy me a new one: an IBM PC with whopping 512 KB of RAM and a Hercules graphics card that was capable of putting two colors on the screen: black and white. That was a serious computer. The lack of games bundled with it might be something that finally convinced my father to buy it and not the much cooler Amiga.

Virtually everyone who owned an Amiga or an ATARI computer in my class made fun of me until I agreed to a bet: that I was able to prove that my computer was just as capable of doing interesting graphics stuff (with fewer colors, granted).

I came home that day and started programming. I had a BASIC dialect installed on my machine, but this proved to be insufficient. I switched to Turbo Pascal (I loved it), but still... At the end I had taught myself x86 assembler and had written my very first demo, a so-called “sinusscroller,” which was a line of animated text that following a sine curve pattern. I won my bet after 3 weeks of programming and about 1500 lines of code. and from that day on I have never stopped programming.

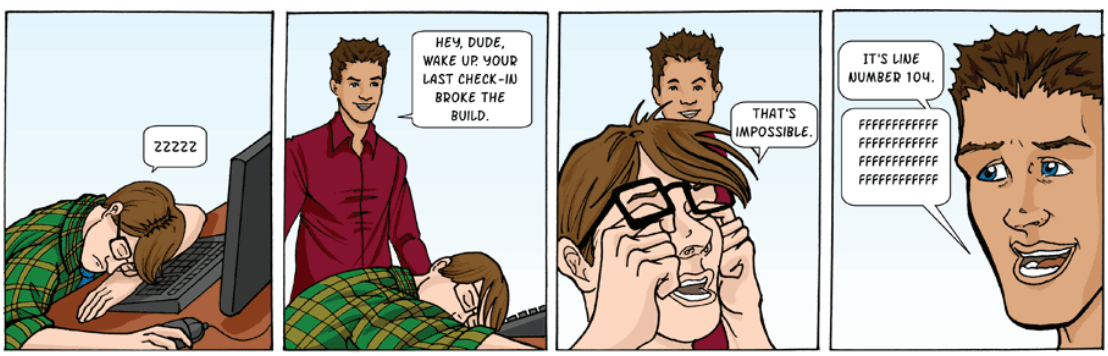
I sat down to write a little sinusscroller while composing this preface. It took me 30 minutes and about 20 lines of code. And - hell - how much fun that was! How powerful today’s frameworks are and uncountable places to implement them!

At home I run a Media Center for which I recently wrote an add-on. It took me ***one day***. I own a smart phone for which I downloaded a homebrew RSS reader. Sitting on top of services provided by the phone's operating system as well as some services in the cloud, it is able to sync via the Internet with my home computer and my Media Center add-on. It amazes me how little effort is necessary in today's IT world to accomplish tasks that would have taken months when I started with my ZX81.

All those frameworks, all those applications and technologies that transform the way we communicate and live today were developed and enhanced by programmers like you and me. What was a sport for nerds and engineers 20 years ago is technology that shapes everyday life and indeed shapes IT.

The {You Shape} It "online magazine" was created for the everyday developer who searches for solutions, ideas, and last but not least, likes to have some fun every now and then. For those developers we have assembled a lot of great material here, from real world tool & coding tips to interviews and articles about today's challenges to outlooks into the future of programming.

{You Shape} It online and this book takes you on a little journey with lot of gems on the way. Enjoy the ride!



a { You Shape } It creation microsoft.com/youshapeit/msdn (cc) BY-NC-ND

Feature Articles

Working with Brownfield Code

by Donald Belcham

During our careers as developers we sit in training courses, conference sessions and vendor demos in which we hear about how easy and great technology XYZ is for creating applications. And almost every one of them assumes that we're working on new development. I'm not sure about you, but I find that new (or greenfield) application development has been fairly rare in my career.

More often than not, we developers are working in companies that already have a suite of custom software. If you're brought in to work on a specific project, it's rare that the project hasn't had something already started on it. As a result, most developers spend a large part of their careers working with or on someone else's code. I don't think this is a bad thing. Instead I think that it is something that we should embrace with good practices, strong tooling and a strong desire to succeed.

If these projects aren't classified as greenfield, how should we refer to them? One of the most common terms in application development is 'legacy'. In his highly regarded book "Working Effectively with Legacy Code", Michael Feathers defines legacy code as:

> *"…code from the past maintained because it works"*

I completely agree with this definition, but it does leave us with a gap. Partially completed applications fall squarely in between legacy (maintained and working) and greenfield (not inhibited by past technical decisions or debt). They have incurred some amount of technical debt and are probably constrained by some design or architectural decisions. They also are only partially completed so we're not working with the code base solely to maintain an already functioning

application. Heck, none of the application may function at all yet. It's these applications that we developers often find ourselves working on.

What is Brownfield?

Brownfield is an increasingly common term being used to define application code bases that are neither greenfield nor legacy. My definition of brownfield is:

> *" a project, or codebase, that was previously created and may be contaminated by poor practices, structure, and design but has the potential to be revived through comprehensive and directed refactoring".*

Brownfield applications exist all over the world, at all companies and in most project teams. Unfortunately they're the norm, not the exception in our industry. The developer across the world, sitting next to you, and even you and I are all creating brownfield applications on a daily basis.

The fact is that as soon as we write a line of code, we have accepted that we are going to maintain it in that form. No matter how well you write code, you are, at the point of finishing that line of code, incurring some level of technical debt and some level of technical constraint going forward. Every single line of code you write is adding to or otherwise modifying the level at which those debts exist.

These statements shouldn't scare you. With today's tools (Visual Studio, Resharper, CodeRush and others) modifying code after it has been created is easier than ever. Refactoring code to lessen the debt or constraints that it places on the project and team is something that is actively encouraged in development circles. There are some things that are important to ensuring that you can do this efficiently and effectively.

Confidence

The largest factor that you need to take into account when working with brownfield code is the confidence level that you, your testers and your client will have with the act of modifying code, without breaking functionality. Nothing scares testers and clients more than the prospect of failing regression tests and bugs introduced to already functional code. It's not just frustrating for them; it's also very demoralizing for a development team. As soon as defects are being introduced from what should be innocuous code refactorings, the development team starts to lose confidence in their abilities to manage the code, their abilities to work with the code in its current state, and the their ability to deliver a product to the client. Those things are a well baked recipe for disaster.

I have seen this on projects which I've joined part way through their completion. In one case I was working with a team where the developers and the business analysts would physically break into cold sweats when there was mention of having to alter one component within the application. They did anything that they could to avoid having to fix bugs or rework that code. That was a team that had completely lost its confidence.

There are some tools that you can use to build or maintain a team's confidence. One of them is automated testing. Even if your team is currently confident about its ability to refactor code, creating a suite of unit and/or integration tests will prove to them that their confidence isn't misplaced. When a team is lacking confidence, adding tests prior to making changes and ensuring that they're still passing after the changes have been effected will work to rebuild that confidence.

I know that some of you will now be saying that you don't have the time, resources or budget to build a comprehensive test suite for your brownfield application. You're probably right. Creating tests after the code has been completed is a laborious task, more so if your project has a significant amount of code. Instead of worrying about comprehensive application-wide test suites, let's think about them in a more localized and granular fashion.

If you're looking to rework a small portion of an application to increase its maintainability, work first at creating a good test suite for that small portion of the application. This approach of creating confidence-building tests only for the portions of the application that are going to be changed will allow you to minimize the effort required for creating tests while maximizing the confidence the team has in changes it makes. This approach will possibly never see your application have a comprehensive test suite, but any changes that you have made will be covered by tests.

The advice for creating tests prior to refactoring code also stands true when the team is faced with modifying code to fix defects or to add new functionality. In the case of fixing defects, first creating a test that exposes the bug (and fails as a result of that bug) allows you to have full confidence that you've fixed it. Likewise creating tests that confirm new functionality prior to creating the code to implement those capabilities within the application will give you a strong understanding of when you've completed the required task.

All Shall Build

Confidence in implementing, fixing and refactoring code is only one part of the brownfield project experience. Confidence on projects comes in many forms. For

instance, do you have full confidence that anyone on your project team could correctly compile and deploy your application to any of your testing, staging or production environments? My experience is that there is usually one person on the team who can perform this task and it usually involves a combination of black magic, voodoo and chicken sacrifices.

Why not work towards a scenario where any one of the developers on the team can, and does, regularly create a release package? Traditional methods of compiling applications in the .NET sphere have been limited to "Rebuild All" within the Visual Studio IDE. While there can be some automation included through the use of pre- and post-build events, "Rebuild All" is a largely manual process when you include tasks such as test execution and release package construction. While this can work, it doesn't necessarily offer the consistency, or flexibility, of using a build script. There are many build scripting tools and languages available to .NET developers. Each offers the same fundamental features, but each also offers a twist.

If you create a compilation, testing and release script that is available to every developer on the team, it becomes one of your projects mechanisms for creating confidence in release management. If the script is available to all developers at all times, they should be encouraged, through simple tooling and fast execution, to run it as many times per day as possible. Every time that you run a script like this you are testing a technically complex portion of your release management process. If that process is being executed tens of times per day, the confidence level of the team and its management and clients should soar.

Social Considerations

We could continue this article with a long and detailed discussion on the merits of using fundamental OO code practices to ensure that your code is flexible, decoupled and robust. Instead I'd like to finish with a brief discussion on the social aspect of working with brownfield applications.

Joining an already started project can present interesting issues both in personal and team dynamics. We all join a new project or job thinking that we can bring greatness to the team or job. As a result we often head into these new engagements bringing suggestions about changes to techniques, tools or practices that we've had success with in the past. Unfortunately this usually happens without us taking the time to properly and completely assess the current situation. This lack of understanding around past decisions and current situations, combined with the vocal desire to try to improve things, often leads new team members to be seen in a negative light.

When joining a team we also have to be aware that the currently serving team members probably have an emotional attachment to the current code base. While many developers believe that they are able to openly take constructive criticism of their past or present endeavours, often they can't. When suggesting changes to a code base, new team members need to be cognizant of the potential negative impact that they may be inflicting on the team's morale. While this is a delicate situation to have to step around while working to improve code, it is one that is vitally important if the team, as a whole, is going to improve productivity, maintainability and quality going forward. Getting team members to believe in the reasons behind the changes that you're proposing instead of the changes themselves can be an effective technique.

Many projects that we developers encounter should be categorized as brownfield. While each of these projects encounters its own set of unique issues, there are some underlying problems that are seen on many. Projects might suffer from a lack of confidence in code, releases and quality. They can also have severe maintainability issues caused by poor coding practices. Those technical issues can be some of the easiest to solve, but each technical problem can expose a series of potentially debilitating social and team cohesion issues.

Don't let any of these issues scare you though. You're probably already working on a brownfield project and having to deal with many of them. Remember that much of what we do in this industry is deal in areas of confidence between developers and management, developers and clients, and developers amongst themselves.

Have a comment about this article? Post it at ***http://devshaped.com/2009/01/working-with-brownfield-code/****.*

Donald Belcham is an independent contractor in Edmonton Canada who specializes in software development with the .NET platform. He is passionate about agile development practices and solid OO fundamentals, while working to provide clients with software that works for their business. Donald is a Microsoft MVP for C#, a user group leader, and a co-author of the Brownfield Application Development (Manning) book. His thoughts on software development can be found on his blog at ***www.igloocoder.com****.*

Beyond C# and VB

by Ted Neward

Within the .NET community, a low-grade battle rages between two groups, those who use Visual Basic and those who use C#, and never has the world of Computer Science seen a more virulent and angry battle… except maybe for those who argue where the curly-braces should go in C++ and C# code. Not that there's really anything to argue about, anyway; as any right-thinking C# developer knows, they go on the next line. But that's not the point of this article, so we'll move on.

(Besides, I'm right, anyway.)

The problem, of course, is that a viewpoint that limits itself to just those two languages misses out on a much larger world, one which contains a much larger set of tools than just those two languages. And while those two languages, like other object-oriented languages before them, are each genuinely useful languages, the unfortunate fact of each is that they are, at heart, just object-oriented languages. And while that may hardly seem to be a limiting factor, considering how much we've been able to accomplish with those languages to date, much of that viewpoint stems from the fact that most .NET developers have never seen anything else.

You know the old saying: When all you have is a hammer….

Back in 1999 and 2000, during the incubation before Microsoft's first .NET PDC, a curious thing took shape. Microsoft invited a number of researchers and academics to participate in a project called "Project 7": as part of its efforts to ensure that the CLR was not just another proprietary technology, Microsoft wanted to see more than just its own efforts running on this new platform. The CLR was at the heart of it, certainly, but the langauges that developers used

would be of far more lasting concern; while Microsoft was reasonably certain that its two lead players (an adaptation of its classic Visual Basic and the brand-new C#) would be a hit, even back then the lead architects on the CLR had a vision that encompassed a larger viewpoint.

Consider, for example, Microsoft's most recent entry into the CLR family of languages, F# (a.k.a Visual F#). Like C# and VB, F# has a number of object-oriented features baked into it, but, unlike C# and VB, F# is primarily a functional language. The nomenclature here is deliberate: just as an object- oriented language drives its users to think primarily in terms of objects, a functional language drives its users to think primarily in terms of functions. This means that certain kinds of programming–such as programming that deals with mathematics or computations, like what might be seen in simulations, financial forecasting, or even game mechanics–becomes much easier than in a language where functions are almost an afterthought.

At this point, if you're like most .NET developers, particularly those who "grew up" on a steady diet of object-oriented languages, trying to see where a functional approach would be non-detrimental, much less beneficial, is probably somewhat hard. After all, "functions" seems to imply "global functions", and didn't the High Priests of Good Design decry global functions almost two decades ago?

Here's a simple example, drawn from a blog post by Chris Smith, of the F# team: **http://blogs.msdn.com/chrsmith/archive/2008/09/04/simple-f-game-using-wpf.aspx**. In it, Chris creates a simple "artillery game", in which two tanks positioned randomly on a flat plain take turns taking pot shots at one another; the challenge is in selecting the correct angle, velocity, and "shot mass" to reach the other tank before they get you.

In itself, this is a fairly simple (yet addictive) game, but the important part of the story is in its basic construction: like many (if not most) F# projects, while parts of the code are made for a functional language like F#, such as the mathematics involved in calculating the shot itself given the user's inputs, other parts are pretty clearly not functional at all, such as the UI interaction with the Windows Presentation Foundation libraries.

CLR interoperability rides to the rescue. Because the F# language is a CLR-based language, it's trivial to put the actual "guts" (the logic) of the artillery calculations into F# functions, then call them from C# code directly, or even indirectly via WPF data-bindings (as Chris does in his code).

Of course, the C# fanatic will quickly point out that C# has a number of features that make functional programming easier–anonymous methods, lambda expressions, basic delegate support, and so on. In the same vein, C# can be argued to be a dynamic language like Ruby, so long as the C# developer doesn't mind programming via the System.Reflection APIs all the time. A vast gulf lies between what a language "can" support, and what a language "natively encourages".

And herein lies the crux of the situation: every language ever designed, from ancient Lisp to modern C#, was designed with particular ideas and concepts in mind. In other words, like every kind of tool designed throughout history, every programming language has a particular usage model in mind when created, and its feature set is carefully crafted around that usage model.

Consider the aforementioned Lisp, for example. While sporting a syntax that only a mother (or a really dedicated programmer) could love, Lisp maintains a very powerful relationship between code and data: that, at the heart of things, code *is* data, and Lisp programs often manipulate code structures every bit as much as they manipulate user-entered data. This allows Lisp programmers to write layers of abstraction on top of abstractions, eventually creating an entirely new layer on top of the programming language itself. (It's no accident that many of the "domain-specific language" advocates look at Lisp as a shining beacon of things to come.) This means, then, that the business domain can be much more closely expressed in the language, rather than trying to mold the domain into a strictly object-oriented model.

Fortunately, these ideas aren't lying fallow waiting for developers to come to Lisp to pick them up and play. Thanks to the foresight of those Microsoft lead architects and developers back in the early days of the CLR's development, a wide range of languages have appeared on top of the CLR (and even more for the CLR's twin brother, the Java Virtual Machine, thanks to its half-decade greater age), many of which explore these ideas in depth. And because they run on top of the CLR, all of these languages can take advantage of the rich ecosystem of .NET libraries and tools, like the .NET Framework Class Library, ASP.NET, WCF, WPF, Workflow, NHibernate, and so on.

For example, consider the language Boo, described as a "wrist-friendly language for the CLR", derived from the scripting language Python. A dynamic language, Boo ranks alongside IronPython and IronRuby as languages that make certain kinds of tasks far easier to complete because of its essentially scripted nature and lack of need for compile-time type-safety. (In other words, practically speaking,

scripting languages make it easier to create higher-level code.) For that reason alone, Boo (and IronPython and IronRuby) merit investigation.

But Boo also provides the ability to "plug in" to the compiler itself, essentially offering developers the chance to extend the language in the same way that Lisp does: by having a look at the structure of the program after it has been parsed, in what compiler wonks refer to as the "abstract syntax tree", or "AST", but before it has been turned into IL. This gives the Boo programmer an opportunity to edit, validate and modify that AST in various ways, providing a clear "meta-programming" capability to the language that C# and VB both lack. In Boo, this is known as a syntactic macro.

For example, a simple macro that comes with the language is the ability to conditionally process certain snippets of code, depending on whether a particular string is provided during the compilation phase. In other words, this is Boo's equivalent to the #if preprocessor directive from C/C++ or C#. (When Boo is interpreting the code in a more scripting-like fashion, the flag is assumed to be passed in to the Boo engine's command-line.) Its usage looks pretty simple:

```
ifdef "BOO":
        print "woohoo BOO is defined"

print "okthxbye"
```

and the definition of the "ifdef" macro is remarkably simple as well, considering that we're hooking into the compilation process itself:

```
import System
import Boo.Lang.Compiler
import Boo.Lang.Compiler.Ast

macro ifdef:
        if not ifdef.Arguments[0] isa StringLiteralExpression:
                raise ArgumentException("ifdef argument must be a string literal.")
        if Context.Parameters.Defines.ContainsKey((ifdef.Arguments[0] as
StringLiteralExpression).Value):
                return [|
                        $(ifdef.Block)
                |]
```

Similar macros come predefined with the language, such as the "using" macro, which serves the same basic purpose as the "using" keyword does in C#:

```
import System
import Boo.Lang.Compiler
import Boo.Lang.Compiler.Ast

macro using:
        expansion = using.Body
        for expression as Expression in reversed(using.Arguments):
                temp = ReferenceExpression("__using${_context.AllocIndex()}__")
                assignment = [| $temp = $expression as System.IDisposable |].withLexica
lInfoFrom(expression)

                expansion = [|
                        $assignment
                        try:
                                $expansion
                        ensure:
                                if $temp is not null:
                                        $temp.Dispose()
                                        $temp = null
                |]

        return expansion
```

This ability to extend the core language is a powerful feature, and one that shouldn't be taken lightly — consider the pain and suffering that Visual Basic developers went through, waiting for the team at Microsoft to introduce the "using" keyword into their language. Had Visual Basic supported a syntactic macro system like Boo's, developers could have added it themselves, alongside any other interesting and useful macros they'd cooked up along the way. Or, if you don't particularly like the exact semantics of the "using" keyword–I personally have always wanted a diagnostic logging statement in the "finally" block generated by the compiler–then in a macro-based approach, the semantics can change without having to go back to Microsoft to ask for a patch.

The point of all this, in case you were concerned, is not that you need to give up your existing fondness for an object-oriented language; Lord knows, I'm not going to any time soon, either. C# and Visual Basic are both languages that serve a useful purpose, and nobody should be telling you to give up something that serves a useful purpose. (Though you'll never get me to admit it, Perl probably does, too, but let's keep that a secret between us, OK?) In fact, the point is precisely the opposite: to make use of *all* the tools available to you–F#, Boo, Nemerle, C# and/or VB, as the problem dictates.

After all, who wants to own just a hammer? Sounds boring, to me.

Have a comment about this article? Post it at ***http://devshaped.com/2009/03/beyond-vb-and-c/****.*

Ted Neward is a Principal Consultant with ThoughtWorks, an international consultancy developing agile solutions. He is fluent in Java, C#, Scala, F#, C++, and a few other languages, and spends a lot of time studying programming languages, virtual execution engines, and scalable enterprise systems. He currently resides in the Pacific Northwest.

Remaining Valuable to Employers

The following is a transcript of video interviews recorded with six technology leaders in October 2008:

How do developers make themselves more valuable to employers during a downturn?

We asked 6 technology leaders.

Barry: Hi, I'm Barry Gervin from Toronto, Canada. I'm a partner at ObjectSharp Consulting and hire software developers.

Billy: I'm Billy Hollis, Nashville, Tennessee.

Bruce: Bruce Johnson. I am a partner at ObjectSharp Consulting in Toronto, Ontario.

Scott: I'm Scott Howlett from Toronto, Canada.

Adam: Hey, it's Adam Cogan. I run a company called SSW. We're a Gold Partner in Sydney, Australia.

Jonathan: Hi, my name is Jonathan Zuck from the Association for Competitive Technology.

You might find their answers surprising.

Barry: It's one thing to learn all the newest and greatest things, but be able to show some sort of portfolio of stuff that you're working on that is fairly current,

shipping technologies; that you're not dogmatic, but a pragmatic developer. People want to get things done with current shipping technology and it's not always about the fancy bells and whistles. People want to just keep things pragmatic and get the job done and focus on delivering value to their customers.

Billy: Developers these days, I think, one of the most value things they can do is get a feel for the parts of their job that are not really what they necessarily like to do but what adds more value and gives them more visibility within their organizations. They need to talk to users more. They need to have more empathy with their users and with the business leaders and try to see things from their point of view. I think that's something that developers aren't very good at, and if they improve those skills, then they make themselves valuable in ways that, for example, can't be outsourced. You can't outsource empathy to India. That simply doesn't work. I've been reading a lot of books about design and such, and it's kind of affected my thinking in terms of developers needing to engage the right-side of their brain more because there's value to be added there. So unless somebody is just a superstar coder on the pure "let's produce technology" side, then I think that they're probably going to make themselves more valuable and more important in their organization to work on some of those more right-brain skills that are not in the skillset of the typical developer.

Bruce: To make yourself attractive to employers right now, I think probably the biggest thing is to be practical about it. You need to be working on things that are adding value to the employer. Ultimately the things that are new, while they're cool and while developers love them, they have the potential to add risk to what an employer does and risk is not what employers are looking for right now. So you need to be working on pragmatic skills, skills that can be easily applied to revenue-producing things, to productivity-producing things for a particular employer.

Scott: The first thing for developers to think about is really to understand that in technology there is a value chain in the same way that there is in manufacturing. If you're at the end of that value chain, if you're the last piece in the wheel, you're the easiest one to let go when demand is falling. So the first thing is to recognize that there is a value chain and to find ways to work up that value chain, whether that's increasing your design skills, helping out in the sales cycle, learning some new tools or technologies, analyzing the market, whatever it is. But if you're the last spoke in the value chain, you're going to be the easiest one to replace. Two other things I'd mention: the first one is to specialize. The world of the generalist developer, I think, is coming to an end. So it's really important to figure out in the whole technology domain where your specialty is and to be passionate about it - that's the last point I'll make. Don't just pick a specialism because you think

there's a market there. Pick a specialism because you actually like it. When you're talking to your employer about it, be passionate. Let your passion flow through.

Adam: If I was a developer, I would try to make sure that the first thing I get right is communication. You can be a great coder, but you are not rounded until you are a great communicator. A great communicator goes and talks to users. Doesn't just talk to users, takes notes so they can action it. Confirms it in an email in case they still got it wrong. And then goes and actions it. And it's no good just being a great communicator and just talking and doing more talking and no action - you've got to have some action. And then you've got to be visible. So my main points would be: talk to users, find out their pain. Talk to the boss, find out his pain. Simple questions like "What's important to you?" Try to find out what it is, repeat it back, see if you've got it right, go away and see what you can do. And become visible in other ways as well. A blog is very important. Get people's thoughts. Speak to mentors - get a mentor! And then go ahead and action some things and get a clear understanding so you can add more value.

Jonathan: I would say that the number one key to making yourself attractive to potential employers is being able to demonstrate that you have a client focus rather than a technology focus. All too often developers are focused on the newest / greatest technology, being bleeding edge in terms of the things they are trying to implement. But if you're able to demonstrate to an employer that what you understand are the client's needs coming first and that delivering value for those clients is going to be your number one priority, you're going to be the most attractive prospect for any employer.

You can view the original video or leave a comment about these interviews online at ***http://devshaped.com/2009/04/remaining-valuable-to-employers/****.*

All I Wanted Was My Data

by Barry Gervin

Most .NET developers are going to need to access relational data at some point. For somebody approaching .NET for the first time, the data access story in the .NET world can be a little overwhelming to say the least. Existing .NET developers looking to update their data access techniques can face a similar overwhelming experience. This article will help you understand the currently shipping mainstream technologies and how you can choose the one that is right for your particular needs.

There is a wide array of choices from Microsoft as well as from within the .NET ecosystem. There are many considerations to keep in mind when evaluating these technologies such as type safety, developer productivity, maintainability, concurrency, database agnosticism, performance, and scalability. One element that is almost universally accepted as a best (and only) practice these days is optimistic concurrency. Just about every data access technology that you'll be interested in is going to use this concurrency model. That is where the similarities end.

Optimistic Concurrency is about being "optimistic" that we will not run into concurrent write access to records in the database. You could think of this as "you probably aren't going to update this record I'm about to give you, and even if you do, no one else is likely to update it at the same time". In practical terms, this means that I'm going to give you the record and forget I even gave it to you. When you want to update it, instead of just giving me the new values, you may want to let me know what version of the record you were editing so I can make sure somebody else hasn't already changed it since I first gave it to you. Even more practically, this means, in addition to using the primary key in the where clause, also give me the original values of every column (or the ones you really care about). For SQL Server, you can be clever and use a single timestamp column that gets updated every time a record is touched.

Traditional ADO.NET

ADO.NET is a .NET brand name for all things data access from Microsoft. Typically using ADO.NET means using SqlCommand objects. Using a SqlCommand, we can pass a native T-SQL statement directly to the database. This could be a SELECT, INSERT, UPDATE, DELETE, a stored proc, or even a DDL (Data Definition Language) statement. Because ADO.NET doesn't know too much about the database syntax, this statement is simply stored in a SqlCommand as a string and passed off directly to the database. You must use one of the Execute* methods on the SqlCommand to invoke the command. Using ExecuteNonQuery will fire the statement and assume it returns no result sets. This is useful in the Insert, Update, and Delete scenarios. To bring a result set back, we use the ExecuteReader method which will return a DataReader. A DataReader looks a little bit like a cursor which will allow us in .NET code to loop through the result set one row at a time and look at the columns. The DataReader is somewhat stupid in that it doesn't know much about the shape or data types of the results coming back so you must use untyped access to get the columns as such in the code below:

```
string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
SqlConnection myConnection = new SqlConnection(myConnString);
SqlCommand myCommand = new SqlCommand(mySelectQuery,myConnection);
myConnection.Open();
SqlDataReader myReader;
myReader = myCommand.ExecuteReader();

// Always call Read before accessing data.
while (myReader.Read())
{
  Console.WriteLine(myReader.GetInt32(0) + ", " +
                    myReader.GetString(1));
}

// always call Close when done reading.
myReader.Close();

// Close the connection when done with it.
myConnection.Close();
```

It is also important to note that the connection to the database remains open during this looping. You may want to reduce the amount of work you do in the loop to minimize the amount of time the database connection is held open. This will make your application more scalable since database connections are a finite resource.

If you are only returning a single value (like a count or exactly one column from one row) you can use the ExecuteScalar method which is a bit easier to use than a DataReader.

SqlCommands and DataReaders in general let you be very precise about the SQL that is sent to the server and control exactly when the connection is opened and closed. You have to provide a bit more muscle in getting the data out of the SQL and coerced into .NET data types, but for performance sensitive areas of your applications, this could pay off. The other thing to be aware of is that there are a different set of objects (Commands, Connections, DataReaders) for each relational database provider. So if you need to move from SQL Server to ODBC to Oracle, you'll need different objects. Fortunately each of these objects are based on an interface and you can program through the interface by using a DbProviderFactory class to create the concrete types based on configuration data. You will have to be sure to use ANSI compatible SQL in your attempt to be database agnostic, which may run counter to providing high speed data access which often requires developers to use native & unique syntax to a particular relational database engine. Obviously your mileage might vary here.

DataSets, DataTables, Data & Table Adapters

With DataReaders you need to be careful about how long you keep your connection open while iterating. Sometimes it is better to pull back your entire result set into memory, release your connection to the database, and then process the result set afterwards. This is where DataSets can be helpful. If you have a batch processing scenario that requires the entire result set in memory or you need to provide end user editing in a grid control, then DataSets are a good place to hold that data.

One of the primary benefits of a DataSet is that it not only caches the data in an efficient memory structure, but it can provide change tracking (what records were added? Deleted? Changed? What were the original values of each column, etc). DataSets work collaboratively with a SqlDataAdapter object, which is simply a composite of 4 SqlCommands (InsertCommand, SelectCommand, UpdateCommand, and DeleteCommand) mapping to the appropriate CRUD behavior (create, read, update, and delete).

DataSets can contain multiple result sets, each mapped into its own DataTable. Tables can be related to each other with Relations which mimic foreign keys. DataSets take care of master-detail linkages and keep everything "hooked up" when inserting records into parent and child tables with identity columns. Each table will require its own SqlDataAdapter. If you are doing a mix of insert,

updates, and deletes and your database has referential constraints, you'll need to orchestrate the changes, making sure parent records are inserted before children and that deletions happen in the reverse order.

DataSets/DataTables come in two flavors: typed and untyped. Untyped DataSets simply infer the table and column definitions after executing the SqlDataAdapter. Fill method. Accessing the columns must be done similarly to DataReaders using column names in quotes or by positional reference. Typed Datasets on the other hand use wizards and a design surface to generate a strongly typed object. The typed dataset is defined by an XSD document. The XSD is then turned into a strongly typed/named class that essentially sits on top of an untyped dataset. A table mappings collection on the SqlDataAdapter class allows you to use different names for your strongly typed object than that of your database table and column names.

Many people in the .NET 1.0-2.0 era used typed datasets as business entities and wrapped some additional business logic around them. Although you could map column names, you were typically tied tightly to your table design in the database. DataSets can also be converted to and from XML, including a diffgram format that retains pending change information. In many circles, DataSets have and continue to be serialized between distributed layers of an application. This makes for a pretty productive development environment, but has drawn architectural criticisms compared to properly modeled services with messages and contracts.

In .NET 1.0, DataAdapters and DataSets were different objects that really didn't know about each other until you executed the DataAdapter Fill or Update method and passed the DataSet/DataTable to the respective method. It was possible for the table mappings in the SqlDataAdapter to get out of sync with the strongly typed dataset. To improve this situation, Microsoft introduced TableAdapters in .NET 2.0, which are really just DataAdapters that are strongly typed and embedded into the definition of the typed dataset.

Object Relational Mapping

With .NET 3.0/3.5, MS has finally released not one but two ORM toolkits. I say finally because MS previously promised an ORM toolkit to developers in the form of a project called ObjectSpaces. ObjectSpaces was never released and its demise is clouded in a series of potentially bad integrations with WinFS, Project Green, or the MS Business Framework, all of which were eventually cancelled.

The spark that reignited MS ambitions to produce an ORM was the Language Integrated Query (LINQ) project. LINQ's goal as part of C# 3.0 and Visual Basic 9.0 was to provide a set based metaphor for dealing with heterogeneous data types including object collections, XML, and last but not least, relational data. For the relational component of LINQ, the language team came up with a project called LINQ to SQL and shipped it as part of .NET Framework 3.5.

During the same time period, the Data Programmability Group in the SQL Server team had been working on a new layer to sit over top of SQL Server's logical data model. This new "Entity Data Model" (EDM) would allow developers to interact with their relational database using a more abstract conceptual model based on the ideas of Dr. Peter Chen. For this team, the promise of Language Integrated Query was what really brought the notion of the EDM to life, and so began the "Entity Framework" project. The Entity Framework did not ship until several months after LINQ to SQL as part of the .NET 3.5 Service Pack 1 release.

On the outside, LINQ to SQL and the Entity Framework ORMs appear very similar. Both tools use a graphical design tool and wizard to map a relational database into an object model. They can both use LINQ queries to project relational data into objects using composable queries. Both provide change tracking so that changes made to the objects can be persisted back to the database with a single method call.

Given these similarities, Microsoft has spent some considerable energy in explaining when to use each of these technologies. While these two ORMs have strong similarities in programming models, their internal architectures are quite different. These key differences are:

- LINQ to SQL has been locked directly to SQL Server, whereas the Entity Framework includes a provider model that allows it to work with any relational engine. As mid 2009 the list of available providers in the ecosystem included SQL Server, Oracle, Sybase, DB2, Informix, MySQL, PostgresSQL, SQLite, OpenLink Virtuoso, and Firebird.
- LINQ to SQL does not provide a mechanism for streaming records one at a time. Using Entity SQL, the Entity Framework can provide Data Reader style access to results using the same mapping layer used for object queries.
- LINQ to SQL provides an out of the box experience for lazy loading. Lazy loading allows the engine to automatically (by default) go out and retrieve new data as it is needed based on code that access different elements of the object graph. In the Entity Framework, you have to be explicit when you make

trips to the database. Both ORMs allow you to load deep object graphs eagerly as well.

- Perhaps the largest difference between Entity Framework and LINQ to SQL is the conceptual modeling layer that EF provides. EF provides a much richer mapping layer than LINQ to SQL, allowing you to join tables together, provide elaborate class hierarchies with abstract and concrete types, and retain many-to-many relationships in your model. Future plans will see the conceptual model being used in other MS products such as SQL Server Reporting Services and SQL Server Analysis Services.
- Finally, expect to see significant investment in the Entity Framework ORM from Microsoft. Both ORMs are now managed by the same product team at MS and they have stated that while LINQ to SQL will continue to be supported and maintained, future innovation will be focused on the Entity Framework.

Other ORM Technologies

You can't talk about ORMs in the .NET world without mentioning NHibernate. NHibernate or "NH" is a mature yet free open source framework which is a port of the popular Java ORM Hibernate. Perhaps one of the biggest distinctions of NH is its support for a Domain Driven Design workflow which purports a code-first or test-first approach to building applications. Typical NH developers build their classes first and then use an XML file to map those to a database. Unlike NH, both LINQ to SQL and Entity Framework advocate a model-first approach, and one could argue a database driven approach. Both LINQ to SQL and Entity Framework designers assume you have an existing database that needs mapping to some objects that will be derived out of your model.

Although there are other commercial and open source ORM tools available, NHibernate, LINQ to SQL, and Entity Framework are by far the most popular technologies in use today.

Enter Data Services

When it comes to building distributed applications, your mileage will vary with the array of data access technologies. Dealing with serialization and concurrency issues in those environments is beyond the scope of this article. However, it is important to mention what appears to be yet another data access technology available from Microsoft called ADO.NET Data Services, formerly code-named "Astoria". Astoria was made available with .NET 3.5 SP1.

Astoria is an HTTP service layer built on top of WCF that provides a REST-style API to your data, giving each of your elements of data a unique URI (for example, http://host/northwind.svc/Products(1)). Out of the box, this service can be enabled for an Entity Framework model or any other IUpdateable/IQueryable data source in just a few lines of code. Data is queryable and updateable via pure HTTP verbs (PUT, POST, DELETE, and GET) using query strings and HTTP payload. Data can be serialized in either AtomPub or JSON format. The net effect is that your data model is widely interoperable with a dizzying array of potential clients and technologies.

Although you are free to build up complex query strings and HTTP payloads for just about any type of operation, the Astoria team has created client libraries to assist in these endeavors. Firstly, an ASP.NET Ajax library is available on CodePlex to allow JavaScript developers to easily work with Astoria Services. Secondly, as part of the core installation, there is a .NET Client Library which provides a natural query model using LINQ and projecting data into client side .NET objects for use by your .NET or Silverlight projects.

A new project, currently named "RIA Services" (code-named "Alexandria"), builds on top of ADO.NET Data Services by also providing rich validation and UI cues on top of your data model in a client/server model. This technology probably won't be released until around the .NET 4.0 timeframe, but it is definitely something you should keep your eye on.

Application Frameworks

As you can see, the trend in Data Access is to abstract into higher and higher levels within your applications. As data access libraries are generating entity classes, the question around locating data validation logic can become a slippery slope. If a data access generated entity can validate correct data types, maybe it should also validate that postal codes and phone numbers are of the correct format. Should they also cross-validate postal codes with states/provinces? Should these error messages be managed inside of my entities? The waters can be muddied quite quickly.

It is therefore worthy to briefly mention a few of the more popular application frameworks that embrace a holistic view of data access in the scope of an application.

CSLA.NET (Component-based Scalable Logical Architecture) is a framework developed principally by author Rocky Lhotka in conjunction with his popular

series of Business Objects books. When first released, CSLA was focused on Visual Basic 6, but over the years it has remained very current on the latest Microsoft technologies and is now principally maintained in C# and ported to VB. In CSLA, data is fully encapsulated by rich business objects that manage all behavior including persistence. CSLA facilitates the re-use of business objects in many possible client technologies including ASP.NET, WPF, WinForms, and Silverlight including distributed architectures using Web Services.

DevForce from IdeaBlade is a commercially available framework for building rich, distributed applications in Silverlight, WPF, WinForms, and ASP.NET. DevForce builds on top of the Entity Framework for its persistence layer and includes a Business Objects server for distributed architectures. DevForce is a popular application framework and has been around since 2001.

But All I Wanted Was My Data

If you can believe it, this article has been a ***short*** list of the current and popular data access technologies. I think I could safely say that the one piece of universally accepted guidance is that if the data access technology you are considering is not in the above list, then it is likely out dated or obscure and perhaps you should think twice about using it.

If you are comfortable with open source projects and you are a strong believer in Domain-Driven-Development, NHibernate is an obvious choice. There is a strong community to back you up on this decision.

If you prefer to stick with Microsoft Technologies, and you have a legacy database to start from, or prefer a model-driven or data-driven approach, then the Entity Framework is likely where you should end up.

If you prefer to work with a more structured and complete application framework, both CSLA and DevForce are worthy choices at this level and you will have community and paid-support available to back you up on each of these respectively.

The last key piece of advice to offer is that you don't have to pick just one of these. There are pros and cons and you may have to use multiple technologies, even within the same application. If you feel the need for performance, there are no rules against dropping out of your ORM and opting for more direct access. Keep an open mind and use the best tool for the job.

Have a comment about this article? Post it at ***http://devshaped.com/2009/05/all-i-wanted-was-my-data/****.*

Barry Gervin is a founding Partner of ObjectSharp in Toronto, Canada. As a Principal Consultant, Barry provides technical leadership to his valued clients, staff, and the development community. Over his 19 year career in the IT industry, he has led many development teams to successfully deliver large software projects within tight schedules and budgets that consistently perform for their customers. Barry currently serves as a Microsoft Regional Director in Southern Ontario and has received the Microsoft MVP Award for Solutions Architecture for the past 5 years.


THIS REVISED SCHEDULE IS IMPOSSIBLE!
NOT TO WORRY, I SPENT THE WEEKEND CLONING MYSELF.
WHA...?
HEY, I THINK THAT ONE IS PLAYING SOLITAIRE...

MAX(output)

Efficiency Upgrade

by Derek Hatchard

Developers solve problems. Good developers solve problems efficiently. The reward for that efficiency is [insert your preferred reward here]. Let's assume you're simply motivated by the satisfaction of a job well done (or maybe a promotion, a raise, longer lunch breaks, or just some extra time with your family). Whatever the motivation, the secret to solving problems efficiently is not inventing everything from scratch. Certainly the secret is not spinning your mental wheels hoping for inspiration. And most definitely the secret is not using Intellisense to arbitrarily try classes, methods, and properties until something works.

The secret to solving development problems efficiently is having the right information and the means to find the information you don't have.

If you have been developing software for more than a week, you know that a working knowledge of language syntax and program construction (that is, "programming") is only a small part of the necessary skill set for solving problems with software. As a developer, you need to understand the technologies, processes, and methods available to you. You also need to have a set of reliable resources you can turn to when you encounter problems.

Below you will find some resource recommendations to help you find the information you need to help solve development problems more efficiently so you can get on with living the rest of your life.

Finding Answers to Specific Problems

Search Engines

I have heard colleagues muse about the changing role of "software developer," postulating that development has become a job mostly concerned with research and discovery. While that is no doubt an overstatement, the ability to find information quickly is an important skill. The place many of us turn to first is our search engine of choice whether it's Bing (replaces Live Search), Google, or a meta search site like Dogpile.

The appeal of most search engines is the beautiful simplicity of a single search box, but the needs of a developer are not simple when you factor in multiple versions of frameworks, programming languages with similar constructs, and behavior differences between competing products (I'm looking at you, Structured Query Language and Cascading Style Sheets). With just a short time investment, you can learn some advanced search operators and tricks to help narrow search results to exactly the information you need. For example, search for "live advanced search keywords" or "google advanced operators" and you'll find a bunch of operators to help you become a search ninja.

In addition to your search engine of choice, Wikipedia can be an invaluable problem solving tool. I have found Wikipedia to be especially useful when searching for general topics such as common computer science algorithms and concepts. Although I often end up at Wikipedia through search engine results, Wikipedia has its own search box right on the home page (**http://wikipedia.org/**).

Of course not all problems will have solutions waiting to be discovered via a search engine. Some problems have already been solved in another part of your organization and you simply have no idea about the pre-existing solution. There are many tools designed to help companies deal with code asset management and search behind the firewall. One interesting product is Krugle, a specialty search engine built for indexing and searching code.

Stack Overflow

Searching is great when you know exactly what to search for and somebody else has previously solved a similar problem. When you are dealing with a more difficult issue, you may want to pose your specific problem to other developers for advice. There are plenty of discussion boards and forums for developers on the Internet to ask and answer questions.

One intriguing and useful site is Stack Overflow, a "collaboratively edited question and answer site for programmers." This innovative developer Q&A site allows users to post responses to questions and vote answers up or down based on usefulness, accuracy, etc. In addition to posting questions, you can search Stack Overflow or browse posted questions by tag. When you are posting a question on Stack Overflow, the application automatically searches existing questions and shows you matches that might already contain the answer you seek.

For obscure topics and scenarios such as bizarre error codes, Stack Overflow can be a job saver.

Twitter

Microblogging seems to be all the rage these days and some developers find services like Twitter to be an extremely effective way to share tips and send out requests for help. Twitter and similar services can be a two-edged sword, though, as they can be as much a distraction as a tool for information exchange.

If you are not familiar with Twitter, it lets you follow posts or "tweets" from other Twitter users. Tweets can be no more than 140 characters. You choose who you want to follow.

There are plenty of ways to use Twitter that will surely reduce your efficiency, but with a bit of intentionality in your approach, you can build a reasonable size network of trusted colleagues with whom you can swap questions, answers, tips, and tricks. (When your friends hear you are "on Twitter," they may demand to know your Twitter username. You might need to maintain separate Twitter accounts for work and personal use.)

Officemates, Coworkers, and IM Contacts

If you are anything like me, you find it tempting to just ask an officemate or IM contact for an answer. Of course this means solving my problem at the expense of my colleague who is distracted from doing his/her own work. With your recently acquired ninja search engine skills, take a look at some of the research on the impact of interruptions on productivity and ponder how a work culture that accepts interruptions as common practice could ultimately make you a less effective developer. Just something to think about before you start picking the brains of your cubicle mates.

Other Options

What sources do you rely on for finding answers to specific problems? Leave a comment at **http://devshaped.com/2009/05/efficiency-upgrade/**.

Keeping Up with New Technologies

With the rapid and continuous rate of change in the software development space, keeping up with the latest and greatest technologies could be a full-time job – in fact it is in many organizations! It really has become impossible to keep up with everything, but you should not let the fear of information overload prevent you from staying informed about significant changes directly related to your area of expertise. Being informed about new technologies helps you make informed choices about the tools, platforms, and libraries you will use to get from problem to solution.

Blogs

One easy way to stay informed about topics that might affect you is to subscribe to at least a few blogs in your area. Scott Guthrie's blog is a good choice if you only want to get the big headline news like a new release of the .NET Framework (**http://weblogs.asp.net/scottgu/**). If you're a Silverlight developer, Tim Heuer's blog would be a good choice (**http://timheuer.com/blog/**). If you're an ASP.NET developer, perhaps try the Visual Web Developer Team Blog (**http://blogs.msdn.com/webdevtools/**).

If checking blogs via a blog reader is not your thing, you might want to use a service that will deliver new blog posts to you via email. FeedBlitz (**http://www.feedblitz.com/**) is a service I use for several blogs that I want to track even when I'm too busy to spend time reading my larger blog subscription list. Or you might want to add a few feeds to a customized browser home page through a service like Pageflakes or Netvibes.

Podcasts

Another way to stay current on technology is by listening to audio podcasts like .NET Rocks (**http://dotnetrocks.com/**) and Hanselminutes (**http://hanselminutes.com/**). Internet audio shows like these are generally packed with great technical content that helps keep you up to date. Podcasts can be an easy way to transform a tedious commute into a serious professional development opportunity. Personally I use the Zune desktop software (**http://zune.net/**) to automatically download new podcast episodes and sync them to my Zune device. If you don't

have a Zune, you can still grab the MP3 files from the Zune download folder and copy them to any portable media player.

User Groups

User groups are another great way to stay current on new technologies. User groups are community-run technical groups that hold events (both in-person and virtual) for members to learn about technologies related to the focus area(s) of the groups. User group events often happen outside of normal work hours, which can admittedly make it difficult for some people to attend. In addition to learning opportunities, user group events can be a great place to discuss technical issues you are tackling and increase the size of your network of colleagues that you can turn to for help.

Go to **http://www.ineta.org/** to find details (including meeting times and places) for .NET-focused user groups in your neck of the woods.

Challenging Your Thinking

When building software, your existing methods for thinking about problems are used to design a solution based on the tools and information you have available. Too many times I have seen developers stuck in a mental rut and developing solutions inefficiently because of outdated or misguided thinking about the problem at hand. Case in point: I used to spend far too much time building and deploying simple tools and utilities as compiled executables until I learned to embrace scripting as a fully legitimate form of programming (after having turned my back on my past as a UNIX scripting hack). In part my thinking was influenced by the experiences of other developers who were sharing their experiences online.

If you do not have people challenging your think, I believe you will stagnate as a developer. That means talking to or reading the musings of other developers. Ideally you will find yourself disagreeing with or questioning their thoughts on a regular basis. You aren't being challenged if you only read what likeminded people write.

Below are some sources that I find useful in challenging my thinking and assumptions. They might not work for you, which is fine, but I encourage you to find your own trusted sources that will regularly challenge your thinking.

CodingHorror.com is written by the brilliant and prolific Jeff Atwood. It has been a constant trove of insight and reasoned perspective, ranging from discussions of

technical details to choosing a programming chair. (Jeff is one of the founders of the abovementioned Stack Overflow.)

JoelOnSoftware.com is written by the equally brilliant Joel Spolsky. He is the source of such gems as "The Joel Test: 12 Steps to Better Code" and "Things You Should Never Do." In addition to development topics, Joel writes about management and business issues. (Joel is the other founder of Stack Overflow. Hmmm… perhaps I have a bias.)

Ted Neward writes a technical blog (**http://blogs.tedneward.com/**) that I regularly enjoy. Although I don't read every post, I always find his pontifications worth digesting.

Daniel Crenna (**http://dimebrain.com/blog**) is another technical blogger whose musings I enjoy reading. Daniel is a great community champion and the founder of several open source projects. His blogging pace has slowed down in 2009 as he's been working on some useful projects, but I keep hoping he'll starting writing again soon. He's a web / RIA guy, an area I'm interested in, so he stays in the ol' blog list.

Join the Conversation

What do you do to be more efficient as a developer? What are your favorite resources for challenging your thinking? How do you keep up on technology changes in your area of expertise? How do you find solutions to specific problems? Share YOUR tips at **http://devshaped.com/2009/05/efficiency-upgrade/**.

*Derek Hatchard is the founder of Crowd Space (**http://crowdspace.net**). You can find him online at **http://derekhat.com**, **http://ardentdev.com**, and **http://twitter.com/derekhat**.*

Getting Started with Continuous Integration

by Sondre Bjellås

Continuous Integration is a development practice that can help improve your company's ability to deliver quality software. It can reduce the time to market and the deployment cycle from functional complete products to having the solutions deployed on desktop or servers.

Continuous Integration, or CI, can start with a lone developer desktop and mature closer towards production. The ultimate goal is to have a CI environment that can take you from checked-in code to production ready installation packages.

The instructions in this article will help you get started with CI even if you're a single developer or a small team.

This is a process that takes time, but don't be afraid of the road ahead, I will walk you through the first baby-steps towards a well-tuned machine. CI-practice is one of many important elements for the software development industry to mature to levels of other types of manufacturing.

Introduction to Continuous Integration

Martin Fowler has written a very good introduction to CI which you can read this on his website: **http://martinfowler.com/articles/continuousIntegration.html**

> *Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to*

develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage.

The clean and simple essence of CI is to avoid the situation "It works on my machine". This has caused a lot of pain between developer and users, but also between developers. Most of us have experienced that our projects stops compiling properly after a Get Latest operation on the source control system. Another important aspect of CI is to have a cycles of automatic compilation and testing of your projects. We will get deeper into automatic testing later in this article.

Source Control Systems

If you are not familiar with source control systems (SCM), I suggest reading a good introduction by Eric Sink. While the first CI example I will give in this article doesn't use SCM, it is an essential piece of the puzzle if you're doing computer software development. Even as a single developer, it's advisable to use a SCM. SCM will give you better control and management of your projects, with history and the ability to restore deleted code.

Introduction to SCM: **http://www.ericsink.com/scm/scm_intro.html**

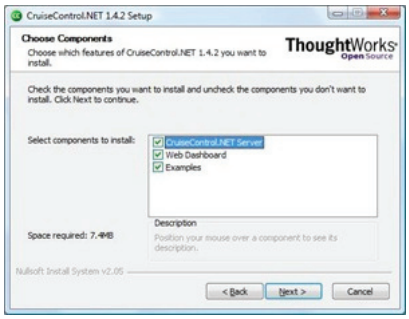
There are many providers of SCM software; for the purpose of this article we will use Microsoft Visual SourceSafe. When your team grows and you have the finances to purchase licenses, I suggest upgrading to Microsoft Visual Studio Team System, which is a complete Application Life-Cycle Management (ALM) suite.

Setting up your environment

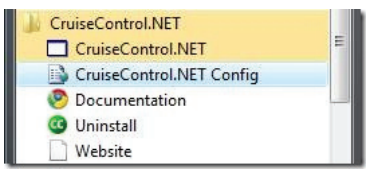
There are various CI solutions available for .NET developers, one of the most popular is Cruise Control .NET (**http://ccnet.thoughtworks.com/**).

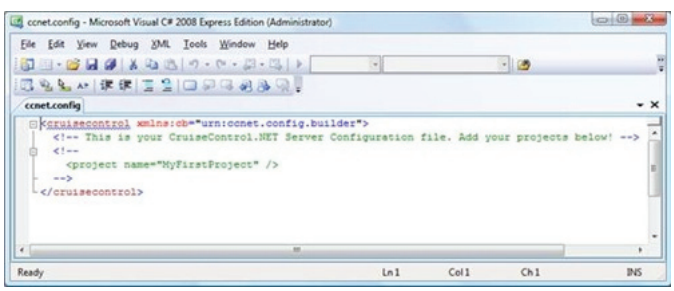
[Please note that all examples are from a 64-bit system where the Program Files paths includes (x86) for 32-bit applications. If you have a 32-bit environment, please make sure you remove any reference to (x86).]

Cruise Control .NET (CC.NET) comes in various forms, for beginners the setup alternative is the simplest to get you started (CruiseControl.NET-1.4.2-Setup.exe).



After the installation has completed, a new folder will appear in the Windows Start menu. From here we can access the main configuration file and launch the console application.





Before we continue with updates to our configuration file, we need to learn a little bit about build tools…

Build Tools

To compile the source code, you will need a tool that does the build for you. One alternative is NAnt, which is a free tool (**http://nant.sourceforge.net/**).

With Visual Studio 2005 Microsoft introduced MSBuild, which is a standalone build engine for .NET. This has made it more accessible for third parties and made it simpler to automate and script.

The simplest way to get started with a build process is to utilize MSBuild, which is already installed if you have .NET Framework 2.0. We will configure CC.NET

to start the MSBuild process which takes care of the compilation of our Visual Studio Solution.

New Project or Existing Solutions

It is much less effort to include guidance, frameworks, and tools in the beginning of a new project. This applies to CI as well. The further you get in the development process, the harder it gets to make everything work properly in a CI-environment.

Sample Projects

For the purpose of this article, we'll create a new Visual Studio solution called *IntegrationSample*. Within this solution we will create 3 projects: *IntegrationSample* (Windows Forms), *Integration.Common* (Class Library) and *Integration.Test* (Class library).

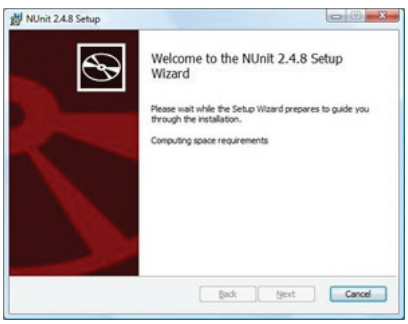
Add a *Person.cs* class to the Common project with the following code:

```
namespace Integration.Common
{
  public class Person
  {
    public string Firstname { get; set; }
    public string Lastname { get; set; }
    public DateTime Birthdate { get; set; }

    public bool IsOlderThan60()
    {
      return (Birthdate < DateTime.Now.AddYears(-60));
    }

    public override string ToString()
    {
      return string.Format("{0} {1}", Firstname, Lastname);
    }
  }
}
```

This class represents part of our business domain model and has certain business logic applied to it that is important to our business, namely the method that tells us if a person is more than 60 years old. The next step is to apply a test for our domain object.

Automated Testing with Unit Tests

Being able to automate the testing process of your projects is an important feature of a CI-environment. Having source code that works as tests for your application logic will help you when you're later going back to modify, extend and fix existing code. With good tests you can ensure that your code changes won't break the whole application. There are different frameworks available for unit testing, the most widely used with .NET is NUnit (**http://www.nunit.org**). To learn more about test-driven development, visit Wikipedia: **http://en.wikipedia.org/wiki/Test-driven_development**.



When the installation of NUnit is complete, you can explore the included samples to learn more about writing unit tests. You can use the NUnit GUI to run your unit tests outside of the CI-environment.

To start writing unit tests, you need to add a reference to the following assembly in your *Integration.Test* project: "C:\Program Files (x86)\NUnit 2.4.8\bin\nunit.framework.dll". Additionally you should create a reference between the Test and the Common project, so we can access our business domain model in the tests project.

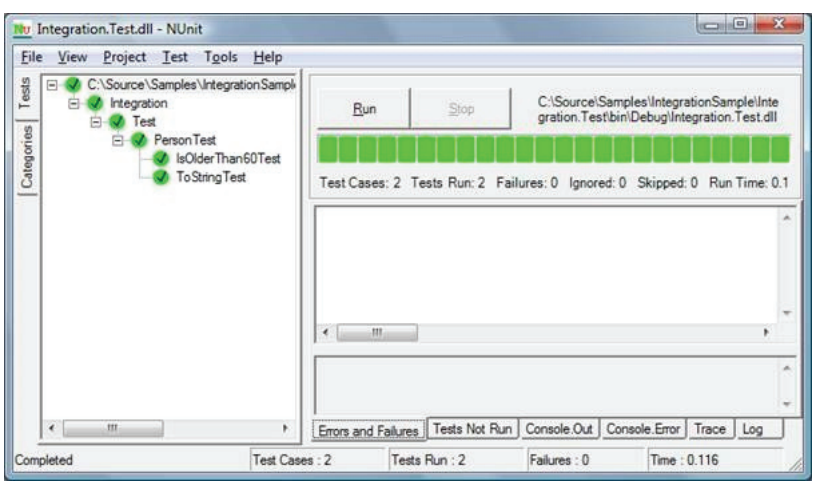
When you're done adding the reference, let's create a class named *PersonTest.cs* which will contain all the unit tests for our specific domain object.

```
using System;
using NUnit.Framework;
using Integration.Common;

namespace Integration.Test
{
  [TestFixture]
  public class PersonTest
  {
    [Test]
    public void IsOlderThan60Test()
    {
```

```
      Person person = new Person();
      person.Firstname = "Sondre";
      person.Lastname = "Bjellås";
      person.Birthdate = new DateTime(1905, 4, 4);
      Assert.IsTrue(person.IsOlderThan60());

      person.Birthdate = DateTime.Now;
      Assert.IsFalse(person.IsOlderThan60());

      person.Birthdate = DateTime.MaxValue;
      Assert.IsFalse(person.IsOlderThan60());
    }

    [Test]
    public void ToStringTest()
    {
      Person person = new Person();
      person.Firstname = "Sondre";
      person.Lastname = "Bjellås";
      Assert.AreEqual(person.ToString(), "Sondre Bjellås");
    }
  }
}
```

Tests are done through assertions. There are many types of assert methods included in the NUnit framework and you should use them to validate your business logic. Writing good unit tests is hard and it's a practice in which you will continually improve as you get more experienced with it.

You can now begin to run and validate the unit tests on your local machine, open the NUnit GUI application and open the Integration.Test.dll file.



As you can see in my screenshot, both of the tests for the Person object validates. When we later go back to change the business logic, we can use the unit tests as a means to validate that our changes doesn't break anything.

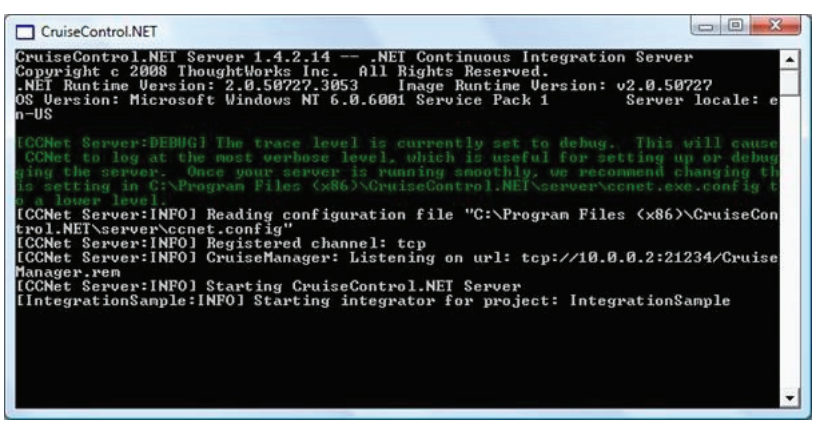
Bringing it all together

Let's go back to the CC.NET configuration file and start modifying it so we get our CI-environment up and running. It's advisable to have a dedicate computer that act as a build-computer, but it's not a requirement.

The first thing we're going to do is modify the default configuration file to run NUnit and our tests, here is the script so far:

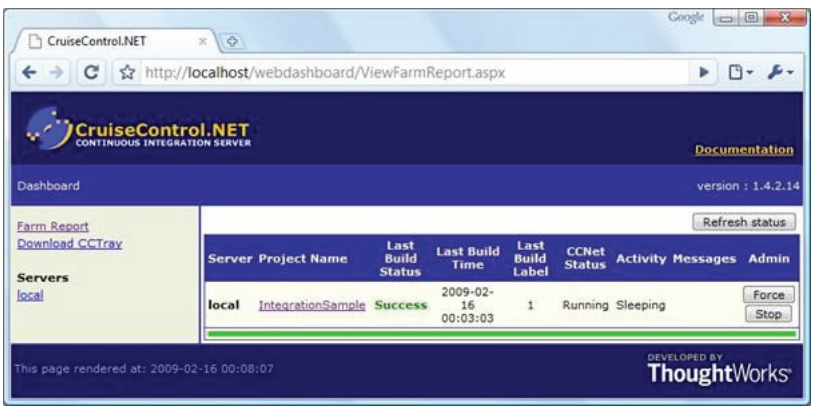
```
<cruisecontrol xmlns:cb=”urn:ccnet.config.builder”>
 <project name=”IntegrationSample”>
  <tasks>
   <nunit>
    <path>C:\Program Files (x86)\NUnit 2.4.8\bin\nunit-console.exe</path>
    <assemblies>
     <assembly>C:\Source\Samples\IntegrationSample\Integration.Test\bin\Debug\
Integration.Test.dll</assembly>
    </assemblies>
   </nunit>
  </tasks>
 </project>
</cruisecontrol>
```

Save the configuration and start the CruiseControl.NET application in the start menu. This will open a console application that will run your configuration. This is useful for beginners and when you're setting up a new environment. When the configuration is complete, you should use the Windows Service for a more stable runtime.



When you ran the CC.NET installation, it should have created a virtual directory for you on the local Internet Information Services. If you happen to have any problems with this, make sure you create a virtual directory called webdashboard and map it to C:\Program Files (x86)\CruiseControl.NET\webdashboard.

Opening up the web dashboard should show us our IntegrationSample project and hopefully a green status, indicating that our unit tests has completed successfully. With the current configuration, we need to click the Force button to run tests.



This dashboard will be your viewport into the status and history of integration builds. You can drill down into the details of a build and you can have multiple CC.NET projects specified in the configuration file.

Our next step is to make CC.NET compile our source code using MSBuild.exe.

Within the <tasks> element in the configuration, add the following script above the NAnt task, which will tell CC.NET to execute MSBuild.exe with our Visual Studio solution file.

```
<msbuild>
 <executable>C:\WINDOWS\Microsoft.NET\Framework\v3.5\MSBuild.exe</executable>
 <workingDirectory>C:\Source\Samples\IntegrationSample\</workingDirectory>
 <projectFile>IntegrationSample.sln</projectFile>
 <buildArgs>/noconsolelogger /p:Configuration=Debug /v:m</buildArgs>
 <targets>Build</targets>
 <timeout>720</timeout>
 <logger>C:\Program Files (x86)\CruiseControl.NET\server\ThoughtWorks.CruiseControl.
MsBuild.dll</logger>
</msbuild>
```

To validate that CC.NET is working correctly, we can go ahead and make a quick modification to the source code. If you reverse the operator for the age method,

we should see the IntegrationSample fail on the dashboard. Drilling down into the details, we should see that our unit test fails.

BUILD FAILED

Project: IntegrationSample
Date of build: 2009-02-16 00:20:54
Running time: 00:00:01
Integration Request: Dashboard triggered a build (ForceBuild)

Modifications since last build (0)

Tests run: 2, Failures: 1, Not run: 0, Time: 0.093 seconds

Failure Integration.Test.PersonTest.IsOlderThan60Test

Unit Test Failure and Error Details (1)

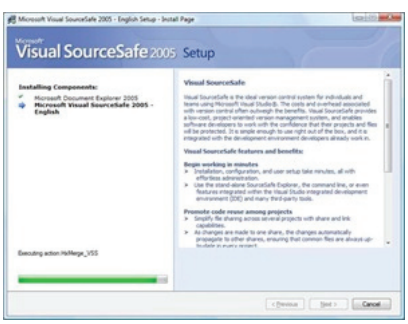
Test: Integration.Test.PersonTest.IsOlderThan60Test
Type: Failure
Message: Expected: True But was: False

at Integration.Test.PersonTest.IsOlderThan60Test() in c:\Source\Samples\IntegrationSample\Integration.Test\PersonTest.cs:line 17

The last and final step is to add support for a source control system. As you have now seen, you can setup an CI-environment without source control, but it makes sense to use source control. A source control system adds history and backup support and the ability to make heavy and local changes and not disrupt the active codebase which is inside the source control database.

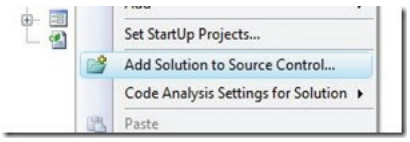
Visual SourceSafe 2005

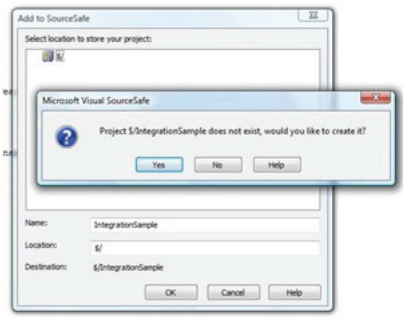
If you don't have a source control system yet, an easy way to get started is with Microsoft Visual SourceSafe 2005. With an MSDN Subscription, you can download it from the downloads page.



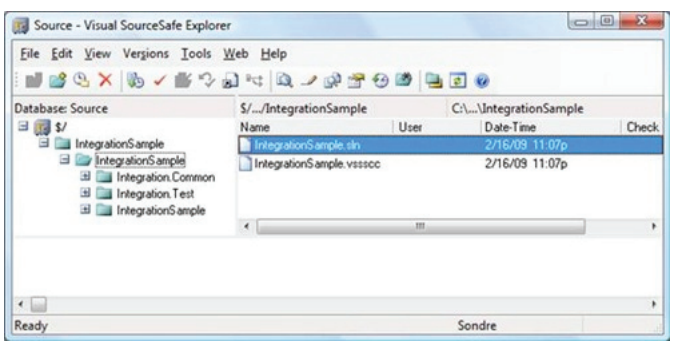
When you go beyond a small team of programmers I suggest upgrading the source control with a more modern provider, like Visual Studio Team System or SourceGear (**http://www.sourcegear.com/**).

Going back to Visual Studio and our IntegrationSample project, we will go ahead and add it to our source control provider of choice.





When the solution and all the projects is added to the source control, we can open the Visual SourceSafe Explorer and see how our newly added project is added and we can begin to check-out and check-in code changes.

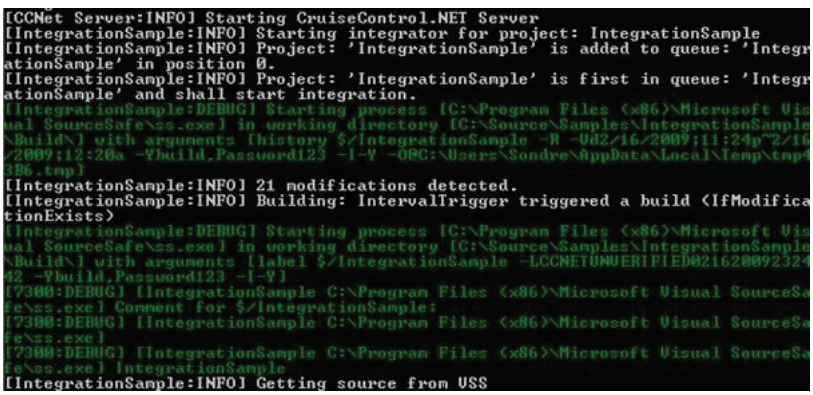


We're now ready for the final changes to our ccnet.config file. Open up the configuration file from the start menu and add the following new elements to the top beneath the <project> element.

```
<triggers>
 <intervalTrigger seconds=”60” />
</triggers>

 <sourcecontrol type=”vss” autoGetSource=”true” applyLabel=”true”>
 <executable>C:\Program Files (x86)\Microsoft Visual SourceSafe\ss.exe</executable>
 <project>$/IntegrationSample</project>
 <username>build</username>
 <password>Password123</password>
 <ssdir>C:\Source\VSS</ssdir> <!-- Path to your SourceSafe database -->
 <workingDirectory>C:\Source\Samples\IntegrationSample\Build\</workingDirectory>
 <cleanCopy>false</cleanCopy>
</sourcecontrol>
```

Make sure you use a different working directory than your normal project folder, if you are running CC.NET on the same machine that you're developing on. You also need to modify the <workingDirectory> and <assembly> elements we created earlier. See the bottom of the article for a full version of our build script.

When you start CruiseControl.NET again, you should see something similar to this output. Please beware that it can take a short while for the process to start.

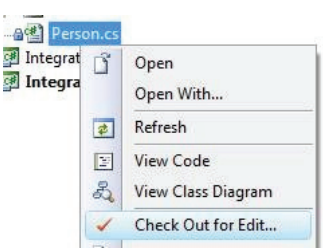


As you can see, the script finds that there has been modification in the project inside our source control, which triggers a get process of the source code and later compilation and test-run.

You can specify your own trigger interval in the configuration file, depending on the size of your development team and your software, you can tune it to make sure you build often, but not so much that your build server starts having trouble.

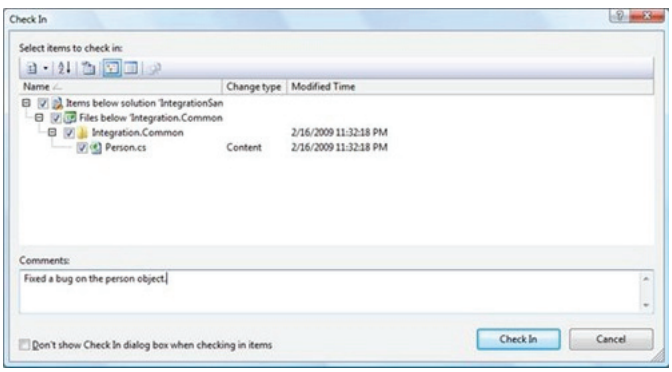
You should create a special user account that your build process can run as and access the source control, as you can see in the example above I created a user called build.

Now it's time to test the whole cycle of editing source code, check-in, watch the build server figure out there has been a modification and trigger a compilation and test-run. Let's check out the Person.cs file and modify the age check to make our unit tests fail.



```
public bool IsOlderThan60()
{
  return (Birthdate > DateTime.Now.AddYears(-60));
}
```

When you're checking code back into the source control system, it's good practice to add an informative comment. This is useful for historical purposes and for other team members to get an overview of changes in the history view.



After checking in our changes, we only need to wait a few seconds to see that CC.NET discovers our change and runs our script.

```
[IntegrationSample:INFO] No modifications detected.
[IntegrationSample:INFO] Project: 'IntegrationSample' is added to queue: 'Integr
ationSample' in position 0.
[IntegrationSample:INFO] Project: 'IntegrationSample' is first in queue: 'Integr
ationSample' and shall start integration.
[IntegrationSample:DEBUG] Starting process [C:\Program Files (x86)\Microsoft Vis
ual SourceSafe\ss.exe] in working directory [C:\Source\Samples\IntegrationSample
\Build\] with arguments [history $/IntegrationSample -R -Vd2/16/2009;11:34p~2/16
/2009;11:24p -Ybuild,Password123 -I-Y -O@C:\Users\Sondre\AppData\Local\Temp\tmpI
E45.tmp]
[IntegrationSample:INFO] 1 modification detected.
[IntegrationSample:INFO] Building: IntervalTrigger triggered a build (IfModifica
tionExists)
```

Congratulations, you now have a CI-environment up and running!

Going Forward

We have looked at the base tools in a CI-environment, going forward I suggest taking a look at frameworks and tools that works good together with CC.NET. FxCop (**http://msdn.microsoft.com/en-us/library/bb429476(VS.80).aspx**) is a tool from Microsoft that analyzes your assemblies and reports anything that is not according to the industry standards for naming conventions, etc. NCover (**http://www.ncover.com/**) is a tool which will generate code coverage reports (that means it will highlight which application logic in your application is being covered by unit tests). If you're a component shop, API documentation is

important. Using tools like SandCastle (**http://www.codeplex.com/Sandcastle**) you can automatically generate help files from your XML comments.

There is a whole world to explore and the extension ability to your CI-environment is endless and everything is there to help you improve the process of building good software with high quality.

You should now understand how to setup your own CI-environment using free tools. Best of luck in setting up your own environment!

Sample Source Code Download

http://devshaped.com/wp-content/uploads/2009/03/IntegrationSample.zip

Build Script

```
<cruisecontrol xmlns:cb=”urn:ccnet.config.builder”>
 <project name=”IntegrationSample”>

  <triggers>
   <intervalTrigger seconds=”60” />
  </triggers>

  <sourcecontrol type=”vss” autoGetSource=”true” applyLabel=”true”>
   <executable>C:\Program Files (x86)\Microsoft Visual SourceSafe\ss.exe</executable>
   <project>$/IntegrationSample</project>
   <username>build</username>
   <password>Password123</password>
   <ssdir>C:\Source\VSS</ssdir> <!-- Path to your SourceSafe database -->
   <workingDirectory>C:\Source\Samples\IntegrationSample\Build\</workingDirectory>
   <cleanCopy>false</cleanCopy>
  </sourcecontrol>

  <tasks>

   <msbuild>
    <executable>C:\WINDOWS\Microsoft.NET\Framework\v3.5\MSBuild.exe</executable>
    <workingDirectory>C:\Source\Samples\IntegrationSample\Build\IntegrationSample\</
workingDirectory>
    <projectFile>IntegrationSample.sln</projectFile>
    <buildArgs>/noconsolelogger /p:Configuration=Debug /v:m</buildArgs>
    <targets>Build</targets>
    <timeout>720</timeout>
    <logger>C:\Program Files (x86)\CruiseControl.NET\server\ThoughtWorks.CruiseControl.
MsBuild.dll</logger>
   </msbuild>

   <nunit>
    <path>C:\Program Files (x86)\NUnit 2.4.8\bin\nunit-console.exe</path>
    <assemblies>
```

```
     <assembly>C:\Source\Samples\IntegrationSample\Build\IntegrationSample\Integration.
Test\bin\Debug\Integration.Test.dll</assembly>
    </assemblies>
   </nunit>

  </tasks>
 </project>
</cruisecontrol>
```

Have a comment about this article? Post it at ***http://devshaped.com/2009/04/getting-started-with-continuous-integration/****.*

Sondre is a Microsoft Regional Director with 11 years of experience in software and web development. He currently works as Technology Leader for the Microsoft department at Capgemini in Oslo, Norway. Sondre is additionally the leader of the Norwegian .NET User Group in Oslo and a speaker at various technology and software development events. Active blogger, writer and technology enthusiast with a special passion for robotics, intelligent systems and open source.

On Strike at the Software Factory

by Daniel Crenna

Since there has been software, there has been the dream of the big green button.

Predictions of the future usually involve elaborate thought-capturing machines that deduce the true desire of an end user and effortlessly create their ideal application, ready for orders, often in the form of a simple form and a big green button promising magic with a label like 'Go'. This is great for the futurist daydreamer, but terrible for the software developer who lives a life of continuous improvement, watching in horror as their profession is reduced to an afterthought somewhere between waking and the business application someone dreamt up before lunch. This is right around the time that the very emotional craftsmanship debate kicks in, and the humans vs. machines themes spring to awkward, robotic life.

Since the term "software factories" was coined in the late 60's by Hitachi, it has been linked inexorably to the manufacturing process it is meant to emulate: people fear that the automation of software components will put developers out of work, or reduce their capacity down to mere assemblers of components rather than authors of code. We hear the buzz in technical communities about the emergence of powerful new software factories, developed by Microsoft and others, and can't help but wonder if the intended goal of these new innovations is exactly along those lines. Yet, the people most involved in the software factory movement see it as just the opposite, a way to harness factory processes simply to meet the surging global demand for software at all, let alone with chairs to spare.

> *"We see a capacity crisis looming. The industry continues to hand-stitch applications distributed over multiple platforms housed by multiple businesses located around the planet, automating business processes like health insurance claim processing and international currency arbitrage,*

> *using strings, integers and line by line conditional logic. Most developers build every application as though it is the first of its kind anywhere.*
>
> *Without significant changes in our methods and practices, global demand for software development and maintenance will vastly exceed the pace at which the industry can deliver in the very near future." (from* ***http://softwarefactories.com****)*

Why should you care about software factories? After all they have that irksome quality of sounding like something meant to replace you. The reality is that software factories are a potential answer to a problem that you live with on a daily basis: change. When a software system changes, it creates risk which often hits your desk in the form of bugs to fix. When businesses change, they adopt new technologies, abandon others, or leverage what they already have in new ways. For developers on the ground, this means you either need to predict the future, or jump hoops in the present to keep up with this change and maintain or acquire the skills that make it work.

The future of computing, arguably, is a repeated pattern from its past: enabling technologies are introduced, adopted, and eventually reach maturity in the wake of the next round of capabilities introduced by innovators. Developers working with bleeding edge technologies knows it's their blood. Developers working on mature or sunset systems feel a little out in the cold when they hear what's on the horizon. The software factory is rising in popularity as a way to preempt the technology adoption cycle by introducing automation where a developer would spend time on low yield tasks, and encapsulating domain knowledge where a developer would need lengthy, dedicated time with domain experts. Today, factories are even used to allow business experts themselves to change core behavior of their software, without interrupting the software development life cycle.

You already understand the value of re-usability (think open source), maintainability (think unit testing), and replaceable architecture (think layers or seams); what's a software factory if only more of the same on a grand, possibly unworkable scale, and what's in it for you? Who cares?

As a developer, you should care because the goal of a software factory is to clear a path for you to get to "the good bits"; the business strategy that changes markets, the social networking feature that changes how people communicate, the custom solution that runs a small business like clockwork. In the end you're remembered for how you solved a problem, not how separated your concerns were. The sooner you can get to writing the code that matters to your customer, the better.

Within Microsoft itself, a quiet revolution is forming in the labs and in upcoming updates to our most central development tools. Let's look at some of the software factories that make up the future of computing on the Microsoft stack.

Factories for Business Problems

Domain Specific Language Tools

http://msdn.microsoft.com/en-us/library/bb126235.aspx

A domain-specific language, or DSL, is all the rage at the moment and is likely to crop up in many new development projects in the near future. The power of a DSL lies in its ability to define a specific domain (such as the shipping business, or indoor pool maintenance) in a small utility language that is either used by a developer to quickly script up new modules for a large software package, or taught to business users so they can adapt their software to meet their changing needs. Want to change the business logic around preferred customer treatment? It's one very human readable line in a script, as opposed to a code change, a check-in, and a redeploy across an organization. A DSL is a software factory because it automates the processes of a specific domain, and the artifacts it creates, in this case scripts for customer treatment behaviors, are as reusable as the language definition itself. Microsoft's Domain Specific Language Tools are a collection of visual designers that let you model a DSL visually, similar to the Visual Studio Class Designer or the LINQ to SQL mapping design surface.

M Grammar

http://msdn.microsoft.com/en-us/library/dd129870.aspx

Destined to ship as a feature of Visual Studio 2010 and available today as a technical preview, Mg is first class support for DSL language design. Originally intended to work with other tools for defining and parsing schema, such as XML, directly to databases, Mg is useful in its own right for declarative development of DSLs and it will become a popular tool and topic for the future of DSL modelling with Microsoft technologies.

A video detailing Mg from Paul Vick, Microsoft Architect:
http://www.informit.com/podcasts/episode.aspx?e=E8D94BA1-1B77-4F4D-99CC-BFCE8CE36489

Factories for Architecture

While a DSL would provide a factory for a generalized language, more robust needs require generalized solutions. Several evolutions of full-scale software factories exist in the Microsoft ecosystem, here are a few established and emerging technologies that will allow you to re-brand, repackage, or reconfigure code you've already written, for customers you will have down the line.

Microsoft Enterprise Library

http://msdn.microsoft.com/en-us/library/cc467894.aspx

Microsoft's Patterns & Practices team developed a true software factory long ago when it introduced the Microsoft Enterprise Library. Designed for composing applications out of blocks of clearly defined enterprise features, MEL reduces the number and scale of repetitive tasks that you face every day on a new software project, such as logging, messaging, services, and to some extent, even the flow and functionality of UI. This library requires a fairly low commitment in terms of education as it builds on existing Microsoft technologies but provides a unified set of repeatable features that every enterprise project needs.

Microsoft Managed Extensibility Framework

http://www.codeplex.com/MEF

A lighter-weight library for building composite applications is available with MEF. Borrowing the term composition from the manufacturing domain, the Managed Extensibility Framework is designed to make it easy for a developer to define application features based on components and configuration rather than reinvent the spinning wheel with one-off software implementations for products that exist in many different instances to satisfy specific customer needs. Still in development, look to MEF to be the application block strategy of the future.

T4 Templates

http://msdn.microsoft.com/en-us/library/bb126445.aspx

Revisiting the manufacturing paradigm in software factories, the end result of any factory process is a collection of artifacts. Without artifacts, there is no output and no value to use once the smokestack clears. Arguably the lynch pin behind all software factory approaches, template generation is the artifact concept that what you define at a high level can be broken down and generated into working source code at a lower level. In fact, the very same template engine behind the DSL tools developed by Microsoft is available for you, today, with Visual Studio T4 templates. After defining a T4 template, dropping them into your Visual Solution

awakens them, using meta-data to create entire frameworks, if necessary. One well-known use of T4 templates is within the SubSonic 3.0 database technology (**http://code.google.com/p/subsonicthree/**), developed by Rob Conery of Microsoft; the entire framework for database persistence is created using T4 templates dynamically built around your database's connection string.

Prism: Composite Application Guidance for WPF and Silverlight
http://www.codeplex.com/CompositeWPF/

Need a software factory for differentiated, rich user experiences with the latest Microsoft stack on desktops and the web? Another software factory making a big splash now and in the next few years will undoubtedly be the evolution of Prism. The future of application interfaces involves dynamically assembling behaviors, interactions, effects, and transitions together while being able to reuse each piece in new views. Composition is the buzz word for this new breed of applications that favor declaring what the UI could do, over coding what the UI must do.

Factories for Code Quality

Five years ago, the idea that code testing tools could "predict", "discover", or "experiment" with source code to find fatal flaws and defend against future quality problems would sound like science fiction, but a new breed of heuristic software factory is emerging from Microsoft Labs in the form of intelligent tools for breaking, and mending, software source code. These will be exciting, cutting-edge projects in your near future, and they're available for preview use right now.

PeX
http://research.microsoft.com/en-us/projects/Pex

The goal of PeX is to automate "white box" unit testing. You already know that unit testing is a safeguard, not a guarantee, of quality programs. You try very hard to achieve 100% code "coverage", or all possible paths through an application hunting for bugs both glaring and devious. PeX promises the precision of a computer with the craftiness of a human when looking for software programming flaws. It won't be a stand-in for writing real unit tests by hand, but will help catch what you miss.

CHESS

http://research.microsoft.com/en-us/projects/Chess

To paraphrase the immortal Wesley in The Princess Bride, "Concurrent programming is painful, Highness. Anyone who says otherwise is selling something". Well, in this case, Microsoft Research is on track to improving detecting threading complications in an automated fashion, and you can try it out for free. CHESS looks for "Heisenbugs", a pet term for uncertain, undefinable, intermittent software defects. You know these as the accounting glitch that only happens on Tuesdays, or the dynamically loaded assembly that crashes every one thousand successful operations. CHESS attempts to bring tooling to finding, and more importantly reproducing the conditions that lead to these strange bugs, so you can squash them.

Conclusion

We're very far from the big green button, and thankfully so, but the next five to ten years of software development will bring increasing factory-mindedness to the tasks that return the least value-to-investment ratio, accelerate the time to working software iterations, and assist human innovation with computer precision. Developers should embrace anything that allows them to move from the menial to the meaningful, and should not fear the software factory. Life is too short for one developer to deliver all of the technology that his or her world demands, but software factories in the hands of skilled developers can; factories are the past, present, and future of computing.

*Have a comment about this article? Post it at **http://devshaped.com/2009/04/on-strike-at-the-software-factory/**.*

*Daniel Crenna is a Microsoft MVP and creator of TweetSharp, an open-source Twitter API development library. Daniel is focused on improving the landscape for .NET developers building social software through founding **www.dimebrain.com**. Daniel was a contributor to the Microsoft AJAX Control Toolkit, and is a Microsoft Certified Professional Developer.*

C# Features You Should Be Using

by Ted Neward

When Microsoft first introduced C# to the masses in 2000, it was, in many respects, a language that anyone who'd ever worked with Java or C++ could pick up, learn, and be a productive developer after only a few hours of study. Granted, there were a few subtleties that took a bit of getting used to, like events, but on the whole, the C# language emerged as a pretty close descendant of its predecessors.

But as C# developed, first in the 2.0 release and then in the 3.0 release, the language took a sharp turn from its ancestral legacy into some strange territory by comparison — first with enumerators in 2.0, then lambda expressions in 3.0 — and a number of .NET developers found themselves longing for "the good old days" of C#, worrying that the language was too complicated and painful to use now. Granted, the LINQ features of C# were widely and warmly received, but some of the underlying features that made LINQ possible were barely understood, and to this day, rarely used outside of LINQ.

It's time to put that trend where it belongs — out to pasture. Let's take a look at the top 6 features of C# that you should be using if you're not already. And while we're at it, let's see if we can spare your fingers some trauma, by reducing the number of keystrokes required to get something done.

Auto-Generated Properties (3.0)

This is an easy one, a natural time-saver. How many times have you written code like the following?

```
public class Person
{
  private string m_FirstName;
  public string FirstName
  {
    get { return m_FirstName; }
    set { m_FirstName = value; }
  }

// …
}
```

Oh, sure, I know, several plugins for Visual Studio will generate those property definitions for you, but let's be honest — any time an IDE feature has to write code for you, that's a bad sign. How much nicer is it to just let the language do that for you (since that's what languages do best):

```
public class Person
{
  public string FirstName { get; set; }
}
```

Fingers feeling less traumatized already? The best part is, there's no difference: The compiler will "cook up" the private field and property "get" and "set" blocks for you, doing exactly what you would have. Incidentally, this also solves the "Should I use the field or the property when I'm inside the class" dilemma, since now the only access is through the property, making the code easier to maintain when you do, finally, have to make the properties more than just get/set.

Object Initializers (3.0)

Along the same lines as auto-generated properties, the object initializer is another way the language saves you some finger trauma. Thanks to the prevalence of properties and dearth of intelligent constructors, a lot of .NET code looks a lot like this:

```
Person p = new Person();
p.FirstName = "Katie";
p.LastName = "Ellison";
p.Gender = Gender.FEMALE;
p.Age = 25;
```

In other words, the "new-then-prepare-before-use" idiom. (Don't feel bad if you've been doing this without even realizing it — lots of C++ and Java code look exactly the same). Fortunately, the object initializer can write all the same code in a slightly terser manner:

```
Person p = new Person { FirstName = "Katie", LastName = "Ellison",
Gender = Gender.FEMALE, Age = 25};
```

Now, granted, it's not a huge savings in terms of keystrokes, but it does help stress the "prepare-before-use" nature that constructors are supposed to infer. It also makes it easier to write data-only objects, such as data transfer objects (DTOs), since now you have one less constructor that needs to be written, at a minimum. In fact, if all the constructor is doing is passing values to those same properties, leave it off altogether. This also helps avoid writing four or five (or more) different constructors to cover all the possible intialization permutations, along the lines of constructors that take "FirstName and LastName", "FirstName, LastName and Age", "FirstName, LastName and Gender", and so on.

Implicitly Typed Local Variables (3.0)

This one is simple: instead of writing:

```
Dictionary<Person, List<Person>> departmentList =
  GetDepartmentList("Accounting");
```

… you can let the compiler figure out what the type of departmentLists is by examination, and not have to worry about getting the declaration exactly right:

```
var departmentList = GetDepartmentList("Accounting");
```

Note that despite the obvious syntactic similarity to JavaScript "var" declarations, this isn't a dynamic object — the language still knows it is a Dictionary generic parameterized on a Person and a List of Persons, and Intellisense will demonstrate that to anybody interested in looking. It's simply a shorthand way to avoid some additional finger trauma.

Static Classes (2.0)

Static classes won't spare your fingers, but they can spare your brain, because a static class is deliberately written such that the consumers of a static class can't create an instance — in other words, a static class is exactly what its name implies, a class filled with nothing but static methods and properties. In other words, this is a perfect place to put those "utility" methods that don't really belong on an object instance:

```
// This isn't really an operation belonging to Persons,
// but it doesn't exactly fit well anywhere else in the
// hierarchy. So we put it here.
public static class Legal
{
  public void Marry(Person spouse1, Person spouse2)
  {
  // …
  }
}
```

Static classes make sure that developers realize that these are methods that aren't intended to be object instances — in a way, it's almost a step backwards to the earlier days of procedural programming, but let's face it, sometimes what you really want is just a global method. The various "math" operations come to mind, for example — it's not really an operation of "Int32" to calculate the sine of itself (but extension methods, next, give you that if you want it).

Extension Methods (3.0)

Extension methods are simultaneously one of the most powerful and most dangerous features of C# 3.0 in that they present the appearance of sliding new methods in on an existing and defined type, even if that type is sealed. (In other words, no inheritance is necessary).

Consider the System.Int32 type. As a mathematical type, it doesn't know how to do much besides the four cardinal operations: add, subtract, multiply, divide, and all that. That's great for simple math, but if you're trying to build something bigger and stronger than that, you don't get a lot of help from it. Granted, the Framework has a number of useful operations on the Math class to do some of that, but particularly for some of the unary operations (like taking the absolute of a given number, Math.Abs()), it feels awkward — it would be nice if that was an operation on System.Int32 natively.

Extension methods allow you to do exactly that:

```
namespace MathExtensions
{
  public static class Int32Operations
  {
    public static int Abs(this int i)
    {
      return Math.Abs(i);
    }
  }
}
```

```
namespace ConsoleApplication1
{
  using MathExtensions;

  class Program
  {
    static void Main(string[] args)
    {
      int i = -12;
      int abs_i = i.Abs();
    }
  }
}
```

The syntax is a little indirect, but there's two main parts. The definition of the extension method itself is fairly straightforward–it must be a static method on a static class, whose first parameter is adorned with the extra modifier "this", indicating the instance (the "this" object) on which the method is operating. Next, at the point of usage, the "using" statement brings the extension methods in that namespace ("MathExtensions") into the compiler's field of vision, so that from that point on, those operations are available on the types specified in the "this" parameter. So, in the case above, Abs() appears on "int"s, but not on longs, decimals, floats or doubles (unless those extension methods were also put into MathExtensions, which wouldn't be a bad idea).

The power of extension methods is pretty obvious, particularly when you consider that you could write an extension method taking a "this" parameter of Object, thus adding a method to the root of the entire hierarchy. Do this with care.

Anonymous Methods (2.0) and Lambda Expressions (3.0)

These two features are really the same thing, an easier way to pass a method (instance or static) as a parameter to a function. In of themselves, it may seem like they're just an easier way to write one-off event handlers, but when combined with enumerators (below), it introduces a new style of programming to C#, that of functional programming, in the same vein as languages like Haskell or ML or their more recent brethren, F# and Scala.

Here's a concrete example: we have a collection of Persons in a List that need to see if their birthday is today. We could, of course, rip through the list using the traditional foreach loop, find the Persons we want to find, then apply the logic (add 1 to their age) to each one, but if this list grows to be thousands of objects long, that could take a while.

Instead, let's apply some anonymous methods and enumerators to do some of this in parallel. (Note that all of this is without having to do anything with F#, the Concurrency Control Runtime, or the Parallel LINQ extensions). We'll have to do a little groundwork first, but in a second it'll be clear why this is a powerful extension to the language.

First, let's create an extension method that will add itself to a List<T> and provide a new Filter function that will take a method (in this case, we'll use the Predicate<T> type already defined in 3.0) that will be used to decide which elements should be included in the results. Second, let's create a second extension method on List<T>s that will take an Action<T> type and apply it to each of the elements in that list:

```
namespace ListExtensions
{
  public static class ListOperations
  {
    public static List<T> Filter<T>(this List<T> list, Predicate<T> filterFunc)
    {
      List<T> results = new List<T>();
      foreach (T it in list)
      {
        if (filterFunc(it))
          results.Add(it);
      }
      return results;
    }

    public static void Process<T>(this List<T> list, Action<T> actionFunc)
    {
      foreach (T it in list)
      {
        actionFunc(it);
      }
    }
  }
}
```

Using the new methods is pretty easy, as they just "appear" on the List<T> type once the ListExtensions namespace has been "using"ed:

```
namespace ConsoleApplication1
{
  using ListExtensions;

  class Person
  {
    public string Name
    {
      get; set;
    }
    public DateTime Birthday
    {
      get; set;
    }
    public int Age
    {
      get; set;
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      List<Person> database = new List<Person>(
        new Person[]
        {
          new Person {Name = “Ted Neward”,
                      Birthday = new DateTime(1971, 2, 7),
                      Age = 38},
          new Person {Name = “Charlotte Neward”,
                      Birthday = new DateTime(1971, 11, 3),
                      Age = 38},
          new Person {Name = “Michael Neward”,
                      Birthday = new DateTime(1993, 7, 3),
                      Age = 15},
          new Person {Name = “Matthew Neward”,
                      Birthday = new DateTime(1999, 9, 3),
                      Age = 8},
          new Person {Name = “Cathi Gero”,
                      Birthday = new DateTime(1971, 4, 3),
                      Age = 38},
          new Person {Name = “Newborn Baby”,
                      Birthday = DateTime.Now,
                      Age = 0}
        });
      database.Process((p) => Console.WriteLine(“{0} is {1}”, p.Name, p.Age));
      Console.WriteLine(“============ Checking for birthdays ==============”);
      database.Filter((p) => p.Birthday.Date == DateTime.Today.Date).Process((it)
                      => it.Age = it.Age + 1);
      database.Process((p) => Console.WriteLine(“{0} is now {1}”, p.Name, p.Age));
    }
  }
}
```

So far, so good. But remember, we started this discussion talking about doing operations in parallel, and these are all happening in sequence. To demonstrate doing them in parallel without changing the basic programming model, let's create a "Parallel Process" function, ParProcess, as an async-delegate-invoking version of the Process function above:

```
namespace ListExtensions
{
  public static class ListOperations
  {
    // …

    public static void Process<T>(this List<T> list, Action<T> actionFunc)
    {
      foreach (T it in list)
      {
        actionFunc(it);
      }
    }
    public static void ParProcess<T>(this List<T> list, Action<T> actionFunc)
    {
      List<IAsyncResult> iars = new List<IAsyncResult>();
      foreach (T it in list)
        iars.Add(actionFunc.BeginInvoke(it, null, actionFunc));

      foreach (var iar in iars)
        ((Action<T>)(iar.AsyncState)).EndInvoke(iar);
    }
  }
}
```

This would be a bit easier to write if we didn't have to call EndInvoke on each async delegate invocation, but CLR lore holds that failing to do so leaks resources (and presumably, after enough of them, will crash the process). Writing an equivalent "Parallel Filter" function is trickier, since each asynchronous invocation will need to add the compared item into a collection to be returned to the user, but it's not impossible — I leave that as an exercise to the reader. (Which is another way of saying, "I could have written this, but I was feeling either lazy or under terrible pressure to turn in my overdue article".)

Using the ParProcess version is no different than the Process version:

```
namespace ConsoleApplication1
{
  using ListExtensions;

  // …

  class Program
  {
    static void Main(string[] args)
    {
      List<Person> database = new List<Person>(
        new Person[]
        {
          new Person {Name = “Ted Neward”,
                      Birthday = new DateTime(1971, 2, 7),
                      Age = 38},
          new Person {Name = “Charlotte Neward”,
                      Birthday = new DateTime(1971, 11, 3),
                      Age = 38},
          new Person {Name = “Michael Neward”,
                      Birthday = new DateTime(1993, 7, 3),
                      Age = 15},
          new Person {Name = “Matthew Neward”,
                      Birthday = new DateTime(1999, 9, 3),
                      Age = 8},
          new Person {Name = “Cathi Gero”,
                      Birthday = new DateTime(1971, 4, 3),
                      Age = 38},
          new Person {Name = “Newborn Baby”,
                      Birthday = DateTime.Now,
                      Age = 0}
        });
      database.Process((p) => Console.WriteLine(“{0} is {1}”, p.Name, p.Age));
      Console.WriteLine(“============ Checking for birthdays ==============”);
      database.Filter((p) => p.Birthday.Date == DateTime.Today.Date).Process((it) =>
it.Age = it.Age + 1);
      database.Process((p) => Console.WriteLine(“{0} is now {1}”, p.Name, p.Age));

      Console.WriteLine(“========== Processing Asynchronously =============”);
      database.ParProcess(delegate (Person p) {
          Console.WriteLine(“Printing {0} on {1}”, p.Name,
          Thread.CurrentThread.ManagedThreadId);
          Thread.Sleep(5*1000); });
    }
  }
}
```

When run, the results that appear for the managed thread id’s may differ, but on my system, they ranged from 3 to 7.

Summary

In some ways, this may seem like a perverse way to take features of a programming language and do crazy things with them — after all, all of the code I've written above could have been done using "traditional" C# / object-oriented approaches, using traditional looping approaches and so on. I don't think any of the designers of C# would hesitate for one moment to agree with that statement.

But the more we as C# developers can look at the new features of the language, including the new "dynamic" features that are coming in C# 4.0, and use them to better and more succinctly express the intent of the code or design, the more we can reduce the amount of code we have to write, ship, and maintain. And in the end, let's be honest — the best line of code we've ever written is the one that we didn't have to write in the first place.

Have a comment about this article? Post it at ***http://devshaped.com/2009/03/c-features-you-should-be-using/****.*

Ted Neward is a Principal Consultant with ThoughtWorks, an international consultancy developing agile solutions. He is fluent in Java, C#, Scala, F#, C++, and a few other languages, and spends a lot of time studying programming languages, virtual execution engines, and scalable enterprise systems. He currently resides in the Pacific Northwest.

Accelerate Your Coding with Code Snippets

by Brian Noyes

One of the most underutilized productivity features in Visual Studio 2005 and Visual Studio 2008 is Code Snippets. Even though most developers have heard about them, I find when working with consulting customers that there are few developers who use them on a regular basis. Even if they do use them, they vastly underutilize them by only using a couple of them and never create their own. In this article, I give you a quick intro into what code snippets are, how they work, what is available out of the box, and how to go beyond that by quickly creating your own code snippets.

Code Snippets Overview

Code snippets were introduced in Visual Studio 2005 and allow you to emit a chunk of code into your editor by typing a few keystrokes (a shortcut mnemonic). I'm going to use C# for the examples in this article, but just realize that code snippets work in Visual Basic and other .NET languages as well. That chunk of code can just be a static chunk of code, but what really makes code snippets powerful is that they can have placeholders that you can quickly overtype when you invoke the code snippet.

To try them out, the best one to start with is one you can probably use the most often: the prop snippet. Code snippets have both a title and a shortcut, although these will often both just be set to the shortcut value. The prop snippet emits a property with a backing member variable in Visual Studio 2005, or an auto-implemented property in Visual Studio 2008. Code snippets show up in the IntelliSense list as well, so if a snippet has a longer shortcut than a few letters, often you can just invoke it by typing a few of the letters and select it out of IntelliSense.

For example, if I go inside a class declaration and type ***prop*** and hit ***Tab*** twice (once to dismiss the IntelliSense list, the second time to invoke the code snippet), I get the following in the editor:

```
public int MyProperty { get; set; }
```

The type and property name will be highlighted in green, indicating that they are the placeholders defined in the snippet. For this snippet, the type is the first placeholder and will be selected so that if you start typing, you will overtype what is there. So for example, say I want a string property. I type ***string*** and hit ***Tab***. Tab moves between the placeholders, and when the focus is placed on the next placeholder, it selects it as well so you can overtype with the value you want. So if I type ***Name*** and hit ***Enter***, I leave the placeholder mode and my changes are accepted. If I keep hitting Tab, it revisits the placeholders in order in case you accidently tabbed past one or changed your mind as you were filling things in.

This one may not be stunning in its time savings, but when you add up the keystrokes (4 + 1 + 6 + 1 + 4 + 1) you have 17 versus the 30 for typing out the whole property yourself. Wham! You just became almost twice as fast at writing code! OK, don't tell your manager just yet, you need to actually get used to doing this all the time first…

Now consider setting a switch case statement for an enumeration with a number of values. For example, say you needed to set up a switch/case on the states for System.Windows.Input.MouseAction. If you had a local variable of that type with something like the following method:

```
public void HandleMouseInput(MouseAction action)
{

}
```

If I go inside that method and type ***switch Tab Tab***, I get the following snippet with a placeholder on the variable ***switch_on***.

```
public void HandleMouseInput(MouseAction action)
{
  switch (switch_on)
  {
    default:
  }
}
```

If I type in ***action*** for the placeholder (the local variable of enum type MouseAction) and hit ***Enter***, the snippet fills in all the enum values as case statements:

```
public void HandleMouseInput(MouseAction action)
{
    switch (action)
    {
        case MouseAction.LeftClick:
            break;
        case MouseAction.LeftDoubleClick:
            break;
        case MouseAction.MiddleClick:
            break;
        case MouseAction.MiddleDoubleClick:
            break;
        case MouseAction.None:
            break;
        case MouseAction.RightClick:
            break;
        case MouseAction.RightDoubleClick:
            break;
        case MouseAction.WheelClick:
            break;
        default:
            break;
    }
}
```

That is a huge keystroke savings filling all the tedious structure code! You just need to fill in what to do for each case.

Built-in Code Snippets

The list of built-in code snippets for C# is actually fairly short. There is really only a small number included out of the box. You can find the full list here with a short description of each for the core C# language snippets: (**http://msdn.microsoft.com/en-us/library/z41h7fat(VS.80).aspx**). There are some additional ones in other categories that ship with Visual Studio 2008, but the list is still fairly short for C#. Visual Basic actually ships with hundreds. But this shouldn't make you feel shorted as a C# developer. You can actually find a greatly expanded library of C# code snippets that shipped after VS 2005 here: **http://msdn.microsoft.com/en-us/vs2005/aa718338.aspx**. You can also find libraries of other code snippets out on CodePlex and other places on the net. ***do***, ***while***, ***for***, ***foreach***, and a number of other common control structures are all in the basic set, along with everything from data access patterns to file IO patterns in the expanded sets. ***ctor*** is another one that is handy for declaring a default constructor for a class.

There is a Code Snippet Manager in the Tools menu of Visual Studio that will let you browse all the built in snippets, import new ones, and search online. However, if you understand what is going on under the covers, it is actually easier to just manipulate them as files.

Inside Code Snippets

Code snippets themselves are nothing more than XML files with a particular schema that reside in some directories that Visual Studio is aware of. Specifically, snippet files end with a (not surprising) extension of .snippet. The built-in code snippets that ship with Visual Studio reside under the folder C:\Program Files\Microsoft Visual Studio 9.0 under VC# or VB respectively. Visual Studio is the default editor for that file type, so you can just browse to those folders and double click on the .snippet files and edit the XML in Visual Studio. If you want to create your own snippets, put them in your Documents folder under Visual Studio 2008\Code Snippets\Visual C#\My Code Snippets (or similar path for VB or 2005). Visual Studio watches this file all the time, so changes or additions to snippets take effect immediately without needing to restart Visual Studio.

There are a lot of elements supported in the schema but the key ones to focus on are Shortcut in the Header element, and the Declarations and Code elements under the Snippet element. Say I wanted to emit a code snippet like the following with a placeholder over the Boo! string so I could overwrite it:

```
System.Console.WriteLine(“Boo!”);
```

To do so, I would have a code snippet that looks like this inside:

```
<?xml version=”1.0” encoding=”utf-8” ?>
<CodeSnippets  xmlns=”http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet”>
  <CodeSnippet Format=”1.0.0”>
    <Header>
      <Shortcut>cwboo</Shortcut>
    </Header>
    <Snippet>
      <Declarations>
        <Literal>
          <ID>output</ID>
          <Default>Boo!</Default>
        </Literal>
      </Declarations>
      <Code Language=”csharp”>
        <![CDATA[System.Console.WriteLine(“$output$”$end$);]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

In this case, the IntelliSense shortcut to invoke it is ***cwboo***. The code snippet declares one literal variable named output with a default value of ***Boo!***. The actual code of the snippet has to be declared inside an XML CDATA section because some legal characters in C #and other languages could cause XML parsing errors. The ***output*** literal is a named placeholder in the emitted code which will be replaced with the default value and highlighted for overtyping as discussed earlier. You can see that when you use a literal in the snippet, you enclose it within leading and trailing $ delimiters. Everything else in the snippet (except other variables) is emitted as plain text in the editor when the snippet is invoked. You can see I am also using a built-in literal ***end***, which specifies where the input cursor is placed when you hit ***Enter*** to leave the placeholder mode of snippets.

Creating Your Own Snippets

By now you are probably emitting some form of "Eeewwww!!!" sound at the thought of editing the XML whenever you want to create your own snippets. The good news is that you don't have to. There are a couple of free tools out there to help you create and edit snippets without ever needing to touch the XML directly. One I have been using for years as a C# guy is Snippy (**http://www.codeplex.com/snippy**). It puts a very simple dialog interface on top of the XML schema for you to create your snippets (see Figure 1).

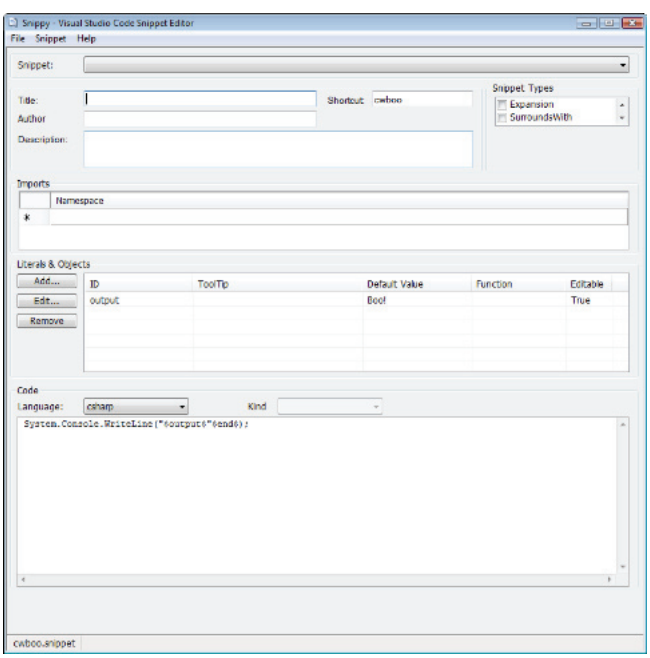


Figure 1: Snippy

A great way to start many custom snippets is to start with an existing one and just edit it. Just remember to do a "Save As…" and save it to your Documents code snippets folder mentioned earlier so you don't overwrite the built-in ones.

Another choice is the Code Snippet Editor for Visual Basic 2008 (**http://msdn.microsoft.com/en-us/vbasic/bb973770.aspx**). For those language bigots, you will have to get over the title: it actually supports multiple languages (see Figure 2).

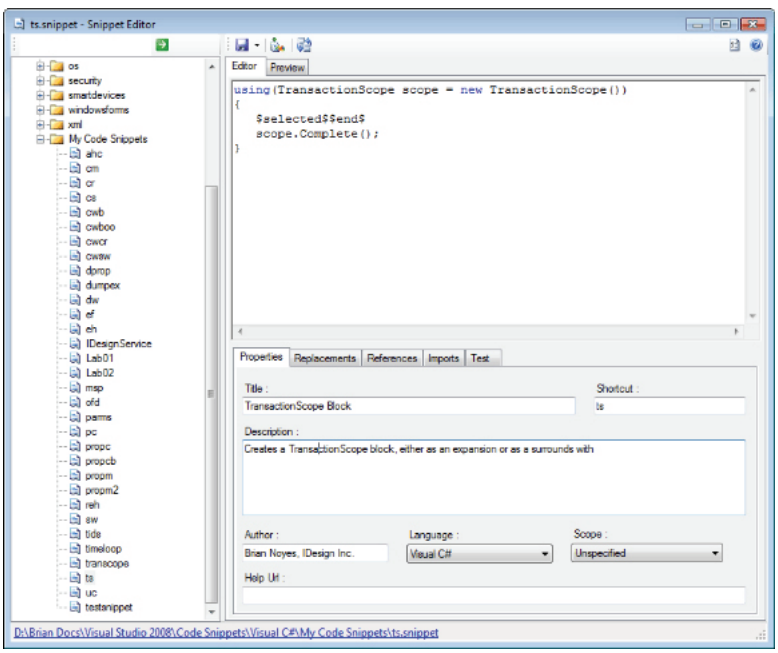


Figure 2: Code Snippet Editor for Visual Basic 2008

Either one of these will allow you to quickly create your own snippets and stop typing the same code over and over again.

Code Snippet Functions

I showed you the switch snippet earlier in the article, and that may lead you down the path of thinking you can pull off anything with a snippet. Unfortunately that is one of the most powerful ones you will find. There is a whopping three functions in the C# library for expanded capabilities in your snippets (**http://msdn.microsoft.com/en-us/library/ms242312.aspx**). The ***GenerateSwitchCases*** function is the one that did the magic for the switch code snippet. The ***ClassName*** function will emit the class name of the class in which the snippet is invoked. This is used by the ***ctor*** snippet I mentioned earlier to declare a default constructor. And lastly there is the ***SimpleTypeName*** function, which will take a fully qualified type name, and will emit both a ***using*** statement

in C# (***Imports*** in VB) and the root class name so that you don't have fully qualified types in the emitted code. This is used by the built-in snippet ***cw***, which emits a Console.WriteLine statement. Take a look at that built-in snippet if you want to see the way it is used. Unfortunately there is no way to add your own custom functions. If you want more power to do more custom things in your snippets, you might want to take a look at third party tools such as Developer Express CodeRush (**http://www.devexpress.com/CodeRush**) or JetBrains ReSharper (**http://www.jetbrains.com/resharper/**). They have things called templates that are like code snippets on steroids and are also extensible.

Wrap Up

Now you should have a good idea what code snippets are, how they work, and how to go beyond what you get out of the box. Let me share with you my approach to code snippets: If I find myself typing some code and I get that "déjà vu" feeling of "I know I've written this little chunk of code before", I stop myself, write it as a code snippet, and then I won't ever have to write it again. Too often developers take the longer path because they think they will be "wasting time" by going and doing something like authoring a code snippet for the code they are writing. Many times, it is not authoring the code snippet that is the real waste of time.

*Have a comment about this article? Post it at **http://devshaped.com/2009/01/accelerate-your-coding-with-code-snippets/**.*

*Brian Noyes is chief architect of IDesign (**www.idesign.net**), a Microsoft Regional Director and MVP, writer and speaker. He is the author of Developing Applications with Windows Workflow Foundation, Smart Client Deployment with ClickOnce, and Data Binding with Windows Forms 2.0. He is also a frequent speaker at conferences worldwide including Microsoft TechEd, DevConnections, VSLive!, DevTeach and others. You can contact him through his blog at **http://briannoyes.net**.*


WITH ALL YOUR CLONES AROUND, THERE'S NOT MUCH LEFT FOR US HERE.
NO PROBLEM! LET'S GO GRAB SOME COFFEE.
I SEE YOUR POINT...

Hello, Rich UI World

Is Silverlight 2 Ready for Business Applications?

by Jonas Follesø

Introduction

With the initial release of Silverlight back in 2007 the focus was on building rich online media experiences with support for high definition video content in the browser. However, there were not many features addressing the needs of developers building data centric line of business applications. With the recent release of Silverlight 2 application developers are asking themselves if Silverlight is ready for business applications. Short answer: definitively yes! With Silverlight 2 you get the power of .NET in the browser, a rich control model, strong data binding support and a solid networking stack with support for WCF, ADO.NET Data Services and REST services. Essentially all the building blocks you need to start developing great Rich Internet Applications that solve business needs.

In this article I will try to highlight some of the strengths of Silverlight 2 as a platform for building data centric business applications, compared to traditional web applications. I will also try to address some of the challenges and concerns developers have around adopting Silverlight 2 today. Some of these concerns are around things like printing, search engine optimization, deep linking and bookmarking. All of these issues can be addressed using various techniques I will touch on in this article.

New Possibilities with Silverlight 2

I figured a good place to start would be by looking at some of the new scenarios enabled by Silverlight 2 that previously would have been impossible or really hard using regular web technologies such as HTML and JavaScript. For most developers, the greatest thing about Silverlight 2 is that you get .NET running in the browser on Windows and OS X (and Linux in the future). You can use

your favorite programming language to build your application, supported by the productivity of Visual Studio 2008. You no longer have to use JavaScript to write code on the client (unless you want to). Tooling support for JavaScript was greatly improved in Visual Studio 2008 but still has a long way to go to match C# and VB.NET. Silverlight 2 supports C# and VB.NET out of the box, and you can also write Silverlight 2 applications using dynamic languages such as IronPython or IronRuby.

Having .NET in the browser lets you use the same skill set and programming languages on the client that you use on the server and provides a solid foundation for coding client side validation, business rules, and application logic.

Offline Storage of Data

Silverlight 2 supports offline storage through Isolated Storage, a protected area on the user's hard drive. Each application gets its own isolated space where it can read and write data. This can help application developers improve performance by serializing reference data to XML or JSON and stored it locally for later use. The next time the user loads the application startup time will be improved as reference data is read locally.

Another scenario where offline storage could improve user experience is by saving uncompleted forms. How many times have you accidently closed the browser before completing a web form or had to come back later to complete a form because you where missing some details? In those scenarios the application could save form values locally as the user types. If the user closes the browser and later comes back to complete the form, the application can read the form values from isolated storage.

Access to Local Files and Improved File Uploading

Security is a top priority in Silverlight 2 and there is no way you can read arbitrary files on the hard drive. However the user can choose to give Silverlight 2 access to a local file using the file open dialog. Unlike the standard HTML file input element, you can access the content of the file passed in locally. This can be used to create improved file uploaders that support multiple files at the same time with accurate progress indicators. Or you could use local file access to import and transform CSV or XML files locally before sending the data back to the server.

Multi Threading, Real-Time Networking, and Cross Domain Services

Silverlight 2 supports multi threaded programming, something currently not available in JavaScript. This allows for more complex work on the client without blocking the user interface. Examples of this could be financial calculators, complex validation logic, or other long running background tasks.

The Silverlight 2 networking stack allows real-time network communication over sockets. The client application can open a connection to the server, and the server can push data back to the client. This enables new scenarios, such as constantly up-to-date stock tickers and real-time monitoring of industrial processes or other physical equipment.

One of the limitations developers often run into when working with AJAX is calling web services from other domains than the one hosting the page. In Silverlight 2 cross domain networking is supported if the service explicitly allows this through a policy file. This enables you to call services such as Flickr, YouTube, and Amazon directly from the browser and even access web services hosted on other machines inside the enterprise.

Rich Control Framework for Great User Experiences

So far I have only covered non-visual aspects of Silverlight 2, and in fact all the features I have mentioned so far you could use in a traditional web application by hosting a non-visual Silverlight 2 application and integrate it using the HTML Bridge. But the place Silverlight 2 really shines is the presentation model. You get a control model similar to the one in WPF, and out of the box Silverlight 2 ships with the main controls you would expect like TextBox, Button, ListBox, ComboBox, DataGrid, Slider, DateTimePicker, and many more.

Having a strong set of controls is a key requirement in any data centric business application, and Microsoft has announced that they will continue to improve the Silverlight 2 control story by releasing additional controls as a Silverlight Toolkit hosted on CodePlex. If the controls from Microsoft do not meet your requirements there is already a large ecosystem of third party controls available.

All Silverlight 2 controls support data binding, making it easy to bind your user interface components against properties on your business objects. This greatly reduces the amount of code you have to write in the UI layer as you no longer have to write the repetitive code of moving values from your textboxes onto your business objects. By applying design patterns like Model-View-ViewModel you

can build user interfaces with literally no code behind, making your application easier to unit test.

Strong data binding support combined with the power of .NET makes it easier to write complex validation rules on the client. This enables you to do more validation on the client side without having to do round-trips to the server to validate user input. This isn't something new to Silverlight 2 and can be achieved using JavaScript. However, writing the validation rules in managed code is easier and more productive compared to JavaScript. In some cases you would also be able to reuse some of your business logic code between server and client. A framework like CSLA.NET can help you architect your application in such a way.

One of the strengths of the Silverlight 2 control model is the separation of visual representation and behavior. Using XAML you can redefine the look and feel of a control without losing its behavior. A slider control to set a visual indicator of air pressure in a tank can be restyled to look like a tank where you can drag content up and down. Checkboxes to set gender can be redesigned to use a male/female icon, but still keep the behavior of a checkbox. Similar things can be achieved using HTML and JavaScript, but Silverlight 2 makes scenarios like this real easy and the styling can be done by a designer without having to change the underlying code using the checkbox or slider control. The clean separation of XAML and interaction logic enables new designer-developer workflows previously hard to implement.

Visualizations and Animations

In addition to powerful controls, the Silverlight 2 presentation model has great support for vector graphics, multimedia, Deep Zoom imagery, and animations. Some might dismiss these features as something not suited for a business application. But who said a business application had to be boring? Using the vector drawing capabilities you can create graphics to visualize your business data in new ways. Already there are multiple charting and gauge controls available. Deep Zoom enables you to view gigapixel images over the Internet, opening up new possibilities for data visualization previously not possible. The Microsoft Health Common User Interface application demonstrates how a radiologist could examine and annotate x-ray imagery remotely using Deep Zoom and Silverlight 2.

Animations, when used appropriately, can greatly enhance the user experience of a traditional business application. Based on selections made by the user, animations can be applied to emphasize certain sections of the page, or to show or hide sections based on the selection. Billy Hollis did a DotNetRocksTV

episode in mid-2008 demonstrating a traditional business application where animations greatly help improve the user experience (**http://dnrtv.com/default.aspx?showID=115**).

WPF Portability

Silverlight 2 uses a subset of the .NET runtime, and the presentation model is based on WPF. This allows for both code and skill reuse between platforms. Silverlight 2 is a web technology and there might be cases where you want a full desktop version of your application. Access to local resources, support for offline access, or integration with Microsoft Office could be reasons to use WPF. If you decide to make a WPF version of your Silverlight 2 application you will be able to reuse most of your business logic code directly. There are some differences in the XAML and control styling model between WPF and Silverlight 2 that you have to take into consideration when porting your user interface. Silverlight 2 uses a new Visual State Manager component to manage the look and feel of a control in different states. A checkbox, for instance, will have one look for the checked state and one look for the unchecked state. In WPF you manage different states using triggers, something currently not supported in Silverlight 2. Microsoft has announced that they will include the Visual State Model to the next version of WPF to align it with Silverlight 2. While waiting for the Visual State Model to be implemented in WPF you can use Microsoft Expression Studio architect John Gossman's implementation (**http://blogs.msdn.com/johngossman/archive/2008/08/08/visualstatemanager-for-desktop-wpf.aspx**). This would help make portability of Silverlight 2 applications to WPF easier.

Since Silverlight 2 and WPF support the same vector graphics you will be able to reuse any graphic assets created in Expression Design or Adobe Illustrator. This is great for logos, icons, or any other illustration your designers would create for your project.

Arguably the most important piece of reusability between Silverlight 2 and WPF is skills / experience. The two platforms have so much in common that if you start out with Silverlight 2, the transition to WPF will be a lot easier. Silverlight 2 has a smaller feature set than WPF, making it a more approachable starting point.

If you want to learn more about WPF portability, I have a blog post describing the process of porting my Dive Log sample application from Silverlight 2 to WPF at **http://jonas.follesoe.no/PortingTheSilverlightDiveLogApplicationToWPF.aspx**.

Deep Linking, Search Engine Optimization, and Printing

One question I get asked a lot is how search engines deal with Silverlight 2 content. From a business application point of view, this might not always be an issue as many applications will be hosted inside the enterprise behind password protected sites. However, there are scenarios where search engine optimization is important for Silverlight 2 business applications. One example could be an online product catalog created in Silverlight 2. I'm sure the marketing department would be rather concerned about the discoverability of their products using search engines. The two key things to think about for search engine optimization are indexability and relevance. Indexability is achieved by addressing what content is visible to the crawler and where the crawler should look. Relevance is achieved by providing deep linking to individual pieces of content so that people can link to your application.

In our product catalog example deep linking could be implemented by adding a parameter to the URL hosting the Silverlight 2 application indicating which product to load. When the Silverlight 2 application loads it could read the URL and load the correct product. This would enable users to share links directly to a specific product or add a bookmark. At the same time the web server rendering the initial page hosting the application could provide a HTML representation of the product page hidden underneath the Silverlight 2 control. The user would only see the Silverlight 2 representation of the product catalog, but the crawler could index the HTML content underneath. To help the crawler find all the products you could supply a sitemap file containing links to each individual product in the catalog. For more information on search engine optimization check out Nikhil Kothari's blog post on Search for Rich Internet Applications: **http://www.nikhilk.net/Search-RIA.aspx**.

Printing in Silverlight 2

By default Internet Explorer 7 supports printing of Silverlight 2 content but other browsers such as Safari and Firefox currently don't. For our product catalog this would be a major issue. Rendering HTML content behind the Silverlight 2 control would not only improve indexability, it could also help us improve the printing experience. Using standard CSS print stylesheets, we could hide the Silverlight 2 container when printing the page and instead apply styles to show the HTML content sitting behind it. When the user browses the catalog and selects new products we could use the HTML Bridge in Silverlight 2 (**http://msdn.microsoft.com/library/cc645076(VS.95).aspx**) to dynamically upgrade the HTML content based on the currently selected product. This way the user always gets a nice looking printout in all browsers. I have a blog post describing this technique as

well as covering printing in Silverlight 2 in depth at **http://jonas.follesoe.no/PrintingInSilverlight2UsingCSSAndASPNETAJAX4.aspx**.

For some business applications with more advanced reporting requirements, CSS print stylesheets might not be enough. In those cases you could use the HTML Bridge to invoke JavaScript to open up a new browser window containing a server side generated report. The report could be any document, such as a PDF or XPS file, or simply an HTML page rendered on the server. Since the rendering is happening on the server you are free to choose any technology or reporting framework to create your reports.

Summary

With the recent release of Silverlight 2, now is a great time to start considering Silverlight 2 for your line of business applications. With .NET running in the browser and a powerful presentation model, you now have the building blocks needed to build compelling Rich Internet Applications for use internally or externally on the Internet. Silverlight 2 is a young piece of technology so best practices and patterns are still emerging. Thankfully the similarity with WPF gives us a great starting point, and already application frameworks like Prism and CSLA.NET are being ported over to Silverlight 2. I also expect Microsoft to start adding value on top of Silverlight 2 in terms of application frameworks, reference implementations, and sample applications.

Innovate with Silverlight 2

by Daniel Crenna

One of the reasons we pick up a new technology is the power it gives us to bring better experiences to our customers, clients, and users. If you're excited about a new way to get the job done, there's value in that, but when you can connect that excitement with the people who rely on your work to get their jobs done, then you really have something worth talking about.

Silverlight 2 is approaching its final release. As Microsoft's latest technology for enabling rich interactive experiences on the web, it provides developers a way to use our existing .NET development skills to deliver more to our users than they expect, in the form of enhanced media, graphics, animation, and design capabilities. As a web developer you're already building great experiences today using ASP.NET, so the focus of this article is to solve one of your daily challenges working in ASP.NET in an effective way with Silverlight 2 that gives a better experience for your users.

The Challenge

If you build ASP.NET web applications today that allow users to upload photos, you know that it's a process of posting back file content, manipulating it on the server, and, if necessary, sending the image back to the user after each request. If you want to give your users the ability to interact with that image in the browser, then you'll face another challenge in leveraging a client-side scripting language to provide the wizardry that makes your images come to life.

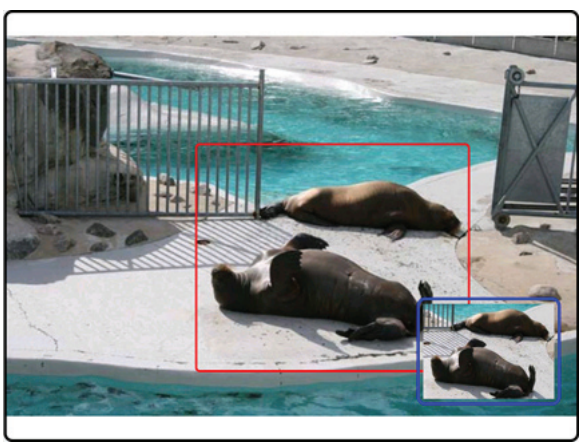
Often your server pays the price, since photos must be uploaded in their entirety to the server before you can take steps to resize or compress the image sufficiently to ease the size of your database storage (and these days, nearly everyone I know owns at least a ten megapixel camera). Of course, you can impose restrictions on

file sizes or provide instructions for your users to crop their images, but you can do better. Silverlight 2 can make the image manipulation story one you can get excited about.

Imagine an application...

Let's think about an application that provides an end-to-end that makes good use of images: a bug tracking application. With this application, a user can spot an issue with a web site, take a screen capture of the problem, and upload it for further analysis. With Silverlight 2 we can provide the user with an interactive experience in which they upload their photo to a Silverlight control, crop the interesting region, and then send the cropped image to the server. You win because only the relevant information is sent to the server, saving bandwidth, and the users win because they don't need any special software or knowledge outside of your bug tracker to get the job done.

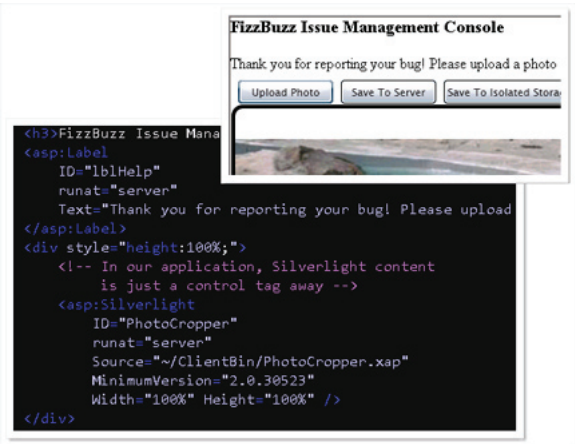
Here's what our bug tracking image component might look like:



Our application has an area to display the working image, and a preview of the cropped selection. The user can click and drag over the working image to define the cropping region.

In with the new but let's keep the old, too

If you want to use Silverlight without throwing away your existing ASP.NET web applications, the good news is that Silverlight can live snugly within an ASP.NET form. This means we can wrap our existing applications around one or several XAML controls that we build in Silverlight. For example, we might decide to use our image component within an ASP.NET page:



You are free to incorporate one or many Silverlight controls in your existing ASP.NET markup. In this example the image cropping component fits neatly within a traditional page layout.

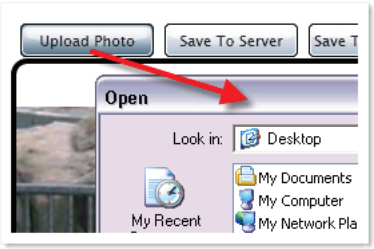
Working with Silverlight 2

To build our application, we'll use XAML, which is a markup language similar to HTML that lets us define our interface. As an example, here is the XAML markup for the main canvas which displays the working image above.

```
<Canvas x:Name="canvas">
    <Border
        CornerRadius="10.0"
        Canvas.Left="1"
        Canvas.Top="1"
        Width="800"
        Height="600"
        BorderThickness="4"
        BorderBrush="Black"
        Opacity="85">
        <Image x:Name="imgUploaded"
           HorizontalAlignment="Stretch"
           VerticalAlignment="Stretch"
           Stretch="Uniform"
           Opacity="85"
           Cursor="Stylus">
        </Image>
    </Border>
</Canvas>
```

XAML gives us an expressive way to define how our application will appear.

As you can see, the thick black border around the working area is defined as a Border element, and the CornerRadius property is responsible for the rounded corners. Inside the 'canvas' element we add an Image control which we can use to display our working image. A similar XAML snippet will define the "picture in picture" preview you see in the application.



Working with Buttons and Events in Silverlight 2 is similar to ASP.NET. You can declare your event handlers, either in XAML or in code-behind, to configure application behavior. In this case we've declared an upload button that will use Silverlight's OpenFileDialog class to select a supported image on the client's computer, open it, and obtain a stream for the selected image. We'll use this stream to display our image in the main form and in the "picture-in-picture" preview area. Silverlight allows immediate use of a user's files using streams, creating opportunities for working with client-side files not previously possible in ASP.NET.

Imagine an interface...

Another nice aspect of Silverlight application development is that we can program against a stateful, persistent UI that simple 'behaves', rather than a UI whose client-side updates require asynchronous feedback to the server (as is the case with ASP.NET AJAX applications) and partial page refreshes to achieve a smooth, snappy experience. This means that to design the interactive cropping region that a user can click and drag around the image, all we need to do is define its appearance in XAML or in code-behind, and change its properties (such as width, height, and location) in order to see those changes come alive in our application. This also means you can perform powerful data-binding scenarios in which you bind your UI elements to values that change in real-time.

*Here we're creating a new **Rectangle** visual element in code-behind and adding event handlers to mouse actions that occur within the main image control. Changing the properties of the '_cropBox' element in response to mouse events is all that's required to achieve an interactive region selection feature that would be difficult to duplicate in ASP.NET.*

Images in Silverlight

Silverlight 2 is designed with a focus on user experience. One aspect of that experience is the size of the Silverlight 2 runtime itself. In order to provide a fast

and lean installation for users, many of the classes you're accustomed to using in .NET and Windows Presentation Foundation (WPF) aren't present in Silverlight. Included in this list are the encoder and decoder classes for image formats like JPEG, PNG, GIF, and BMP.

This means that while Silverlight can effortlessly display PNG and JPEG images that the user selects via the **OpenFileDialog** control, it cannot decode the images into the usable data you need to manipulate them. Fortunately Silverlight is built on a foundation of .NET, which means that whatever functionality we might need for our applications, we can provide our own code to bridge the gap and get the job done. In our case, we're going to draw from a wide variety of community contributions to put together an imaging library that will allow us to easily work with JPEG, PNG, GIF, and BMP files directly in the client application.

Imaging 101

A photo stored in an image format like JPEG is really nothing more than a bunch of bytes that describe the color information, or the color of each pixel in the image. To save space on disk, image formats are compressed, similar to how ASP.NET can compress static resources like CSS and JavaScript files using the .NET 2.0 **GZipStream** or **DeflateStream** classes.

Every pixel color is made up of a combination of values representing the amount of red, green, blue, and alpha, or opacity values. Commonly referred to as RGB, every pixel in an image has all four of these values. As a simple example, an array of bytes that we've obtained from a file stream to a JPEG image, and have successfully decoded, might look like this:

```
[125, 17, 8, 255, 14, 12, 19, 255, 20, 55, 77, 255 …]
```

But what it really means behind the scenes is this:

```
[R, G, B, A, R, G, B, A, R, G, B, A …]
```

This decoded array is very useful because it contains the color information for each pixel in our image. With this information we can reconstruct the image, crop a smaller section of it, apply a filter, or do anything else we desire. Our goal for this application, then, is to take a regular, encoded JPEG, PNG, GIF, or BMP file from a stream we open in Silverlight 2, and translate it into bytes we can manipulate in code.

A community of help

Fortunately, there is an active and vibrant open source community in .NET that we can call upon to solve our challenges. Imaging is a specialized field, which means there are many people who dedicate their careers to producing useful work in imaging for others[1]. While we can certainly learn how image formats work in depth, our real goal is to produce great web applications.

To make use of client-side imaging in Silverlight we're going to draw from several sources to make it happen. One of those sources is Joe Stegman, Group Program Manager of the Silverlight presentation and graphics team. Joe has posted several examples for dynamic image generation in Silverlight[2], several of which we use in our application to provide BMP and GIF decoding support, as well as the ClientImage class that we can interact with to work with decoded images.

We'll still need to provide JPEG and PNG support if we want to round out our image offering and not restrict the user in what kinds of images our application can handle. JPEG support is available from the talented team at Occiptal.com, whose "fluxify" image application allows users to resize JPEG photos on the fly in their browser using Silverlight[3]. They've released the source code for their JPEG decoder, which just leaves PNG support to make our application really shine. Thanks to another open source library, this capability is also part of our cropping application.

Conclusion

If you build web applications, Silverlight can help you build rich, interactive user experiences using skills you already have. You can build Silverlight components that work within your existing ASP.NET applications or build an entire application from the ground up. In this example, we saw how Silverlight makes it possible to implement an image editing feature that is a win for both you and your users: you save server bandwidth by cropping images directly on the client, and users gain a seamless experience working with images within your application. There is a growing developer community around Silverlight and plenty of resources available for you to go even further and discover new ways to impress. Go find out what's possible!

Source Code

You can download the full source code for this article from **http://devshaped.com/files/PhotoCropper.zip**, which goes further to demonstrate the following concepts which you can put to use in your Silverlight applications that use images:

- How to upload images from Silverlight to your ASP.NET server using a WCF service
- How to store the decoded working image in Isolated Storage, a persistent user state capability familiar to .NET Windows client developers

Note: To use the solution, you'll need to install Visual Studio 2008 Standard edition or higher as well as Silverlight Tools for Visual Studio 2008. Choose 'PhotoCropper.aspx' as your startup page in the 'PhotoCropperWeb' web application project within the solution.

Have fun!

References

[1] If you're interested in writing your own image format encoders, like fluxcapacity.net did for JPEG images, you can find a wealth of information online on how each image format is constructed. For example, the JPEG format is described in detail at **http://en.wikipedia.com/wiki/JPEG**.

[2] Joe Stegman's blog (**http://blogs.msdn.com/jstegman**) has several useful posts with full source code, including an EditableImage class that represents decoded or created images you can manipulate directly, as well as example BMP and GIF decoders.

[3] *fluxify* is a great example of client-side image editing in Silverlight 2. Similar to this example, it allows a user to upload a JPEG image and select a pre-defined size, performing all the work on the client side. It is also a good example of the kind of rich interfaces we can build in Silverlight. Check it out at **http://fluxcapacity.net/fluxtools/emailphotos/about.html**.

Licenses and Attributions

PNG Decoder:

Based on the sharppng pr2.code project
http://sourceforge.net/projects/pr2/

PNG Encoder:

Based on the examples written by Joe Stegman

GIF Decoder:

Based on the examples written by Joe Stegman
Derived from the 'GifUtility' project
http://www.codeplex.com/GifUtility

BMP Decoder:

Based on the examples written by Joe Stegman
http://blogs.msdn.com/jstegman/archive/2008/04/21/dynamic-image-generation-in-silverlight.aspx

JPG Decoder:

Copyright (c) 2008 Jeffrey Powers for Fluxcapacity Open Source.
Under the MIT License.

*Daniel Crenna is a Microsoft MVP and creator of TweetSharp, an open-source Twitter API development library. Daniel is focused on improving the landscape for .NET developers building social software through founding **www.dimebrain.com**. Daniel was a contributor to the Microsoft AJAX Control Toolkit, and is a Microsoft Certified Professional Developer.*

Real World WPF: Rich UI + HD

by Gill Cleeren

As a solution architect at my company, I'm responsible for choosing the right technology for the right project. This choice is vital for a project: we must map the customer's requirements to the technology that will solve them in the easiest way. For the customer, this means that no time and no budget is wasted, and that's what really counts.

Several months ago, a special customer request arrived at our doorstep. The customer needed an application built around high definition video playback. Along with the video, several animations had to be synchronized. The final application also needed a very rich user experience that would be easy to work with for inexperienced computer users but look stunning at the same time. Changing the user interface to a new theme would be a nice-to-have.

Working from these initial requirements, several teams made estimates based on different technologies. I was immediately convinced that this project would really benefit from the use of Windows Presentation Foundation and that using WPF I would be able to bring down the total cost for the customer since a lot of the required functionality is available in the platform. We took the dive and proposed a solution based entirely on WPF. And yes, it was retained. The main reasons: time-to-market was lower than other propositions and the demo we prepared with WPF in just one afternoon convinced the customer that the technology was capable of fulfilling the requirements.

Let's go into a little more detail on the specific customer requests…

The problem in more detail

The main goal of the application is displaying training videos of routes. All video content is encoded in high-definition (720p and 1080p). The customer's current platform isn't capable of working with HD content, which is one of the main reasons they needed a new solution.

Along with the high definition content, a map has to keep in sync with the current position in the video, even if the end-user is fast-forwarding or viewing frame-by-frame. The training package also includes a viewer in which the end user can pan and zoom around the map. Finally, a lot of information has to be displayed throughout the application about several points of interest on the map.

Displaying these videos is one thing, creating them is another. An editor had to be created in which it would be possible to build the videos, the maps, and the animations to sync them.

With their current solution, it took the customer about 1 week to create a 10 minute video. We HAD to bring that down!

Incoming: WPF solution

Windows Presentation Foundation proved itself useful in many ways during the development. Working with media, in our case high definition media, has never been easier. High definition is supported out-of-the-box so there was need to buy custom controls. All operations on media like playing, fast forwarding and frame-by-frame support involved writing just one line of code. For example, playing video is simple:

```
mediaElement1.Play();
```

No other platform could have delivered the level of functionality we received from WPF with such ease.

The animations we had to create to have the map follow the video all had to be dynamic. They are created in a custom editor in the application and stored in XML. The player dynamically creates the animations without any performance glitches. Syncing these animations was again nothing more than one line of code (this is getting repetitive, isn't it?). Note that we didn't have to use a single Timer (which is the default reaction developers tend to have when they hear about some time-based actions) for these animations to work - it's supported in the platform.

WPF shines when it comes to creating really good looking user interfaces that can be very flexible when needed. The tools Microsoft provides, including Expression Blend, make the difference here. Although the Expression tools are aimed at the designer audience, they are easily accessible for developers as well.

Compare the relatively bland button on the left with the styled button we created easily with WPF:

Using WPF, it's easy and intuitive to create styles to override the look of several controls. These styles can be made available throughout the application, and when needed, can be changed in one place. This resembles to CSS in web development and makes the XAML code much more readable and maintainable. Updating the application to a new look requires nothing more than changing the styles.

Finally, data binding is one other feature that saved us a lot of time: we had to create several screens that display information. Using the data binding features in WPF, we simply bind our data transfer objects to the UI and WPF does the rest. No more writing tedious code there.

Now that the project is finished, I can confidently say that WPF helped us achieve the customer's goals in much less time than using other platforms. Building the entire application only took us 55 person days. The initial estimates from other teams went well over 100 person days. The platform is ready for business applications and can really be a time saver.

And the developers… What do they think?

One thing I feared was the learning curve for the rest of the team. The project had a strict deadline – the entire project run time was only 2.5 months – and it was the first contact for the other team members with WPF. After some intense reading and coaching, the team was up and running in just one week. They were writing XAML code and designing user interfaces in Blend as if they had been doing it for ages. Apart from some research on specific problems, there were no blocking issues that set us back during the entire development time.

After the final daily Scrum meeting, I asked one developer what he thought about developing in WPF. Here's what he had to say:

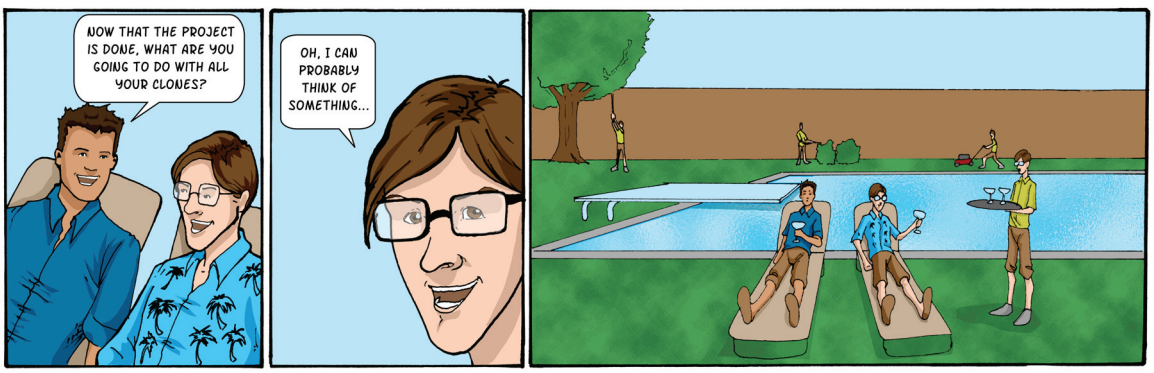
"Making the switch to WPF was very easy. You can leverage your .NET skills entirely, and learning XAML wasn't painful at all. The tools like Blend are developer-friendly and make visually designing the user interface easy."

"I was amazed with the data binding support in WPF that's much more extended than in any other platform. This resulted in repetitive work being less needed so we could focus more on the important aspects of the application."

"I would like another WPF project, please..."

Gill Cleeren is Microsoft Regional Director and MVP (Most Valuable Professional) in Visual Web Developer/ASP.NET. He leads the largest Belgian user group on Visual Studio and Microsoft-related products. In his professional career, Gill is a .NET architect and competence center leader at Ordina.

*You can read more on his blogs at **www.snowball.be** and **www.codeflakes.net**.*


NOW THAT THE PROJECT IS DONE, WHAT ARE YOU GOING TO DO WITH ALL YOUR CLONES?
OH, I CAN PROBABLY THINK OF SOMETHING...

Practical Programming

Hidden Talents

by Peter Jones

The .NET Framework is full of useful functionality, though it's not very useful until you know it's there! Let's take a look at a set of three useful classes / namespaces from the .NET Framework that you probably aren't familiar with (yet).

Let's make some noise: System.Media

Producing sound from a computer is something you may have learned in Programmer Playschool and promptly forgot (like me), but it is a useful little feature to add to all types of software, not just desktop games. Playing a system sound or other sound files is very useful in monitoring environments where the users are not seated in front of a computer 24×7 (for example, in server rooms or manufacturing environments).

System.Media is a namespace that contains three useful classes for playing system sounds and .wav files. System sounds are defined in the Sounds applet of the Windows Control Panel.

What MSDN says:

> *The System.Media namespace contains classes for playing sound files and accessing sounds provided by the system.*
>
> *The following classes are included in this namespace:*
>
> - *A SoundPlayer class for loading and playing sounds in various file formats.*
> - *A SystemSound class for representing and playing a system sound.*

- *A SystemSounds class that retrieves sounds associated with a set of Windows operating system sound-event types.*

Playing a System sound couldn't be easier:

```
SystemSounds.Asterisk.Play();
```

This will play the .wav file associated with the asterisk system sound. SystemSounds is a simple wrapper for the five most common sounds – asterisk, beep, question, exclamation and hand.

If you have your own attention-grabbing, eardrum-busting sound file then you can play this just as easily:

```
string filename = "houston-we-have-a-problem.wav";
SoundPlayer sp = new SoundPlayer(filename);

Loading of large .wav file can take a while so, thoughtfully, Microsoft added support
for loading a sound file asynchronously.
static SoundPlayer _player;

static void Main(string[] args)
{
  string filename = "houston-we-have-a-problem.wav";
  _player = new SoundPlayer();
  _player.LoadCompleted +=
          new AsyncCompletedEventHandler(sp_LoadCompleted);
  _player.SoundLocation = filename;
  _player.LoadAsync();
  Console.ReadLine();
}

static void sp_LoadCompleted(object sender, AsyncCompletedEventArgs e)
{
  if (e.Error == null)
  {
    _player.Play();
  }
  else
  {
    Console.WriteLine("Could not load sound: " + e.Error.Message);
  }
}
```

This simple C# console application will play the sound file after it is loaded. Not only will the loading of the sound occur asynchronously, but the player will also play the .wav file in a new thread.

Additionally, the player can load a wave file from various sources such as a URL, application resources, or a stream.

Brain the size of a planet? MemoryFailPoint

In these multi-gigabyte days it's not often that an application runs out of memory – virtual or otherwise – but there are few things more frustrating to a user than an 'Insufficient Memory' exception. Applications that don't handle these exceptions can potentially corrupt system data (and possibly damage your career!).

Thankfully, there is a simple solution: MemoryFailPoint.

What MSDN says:

> *Check for sufficient memory resources prior to execution.*

MemoryFailPoint throws an InsufficientMemoryException prior to your operation beginning, rather than an OutOfMemoryException at some unpredictable point during the operation. Here's the actual program that is used to launch the Space Shuttle Microsoft (j/k).

```
const int MB_MEMORY_REQUIRED = 2048; // 2 gig

try
{
  securePayload();
  alertMedia();
  loadCrew();
  int countDown = 10;
  beginCountdown();
  while (countDown > 0)
  {
    Console.WriteLine(countDown);
    if (countDown == 3)
      ignition();
    Thread.Sleep(1000);
    countDown-;
  }

  Console.ForegroundColor = ConsoleColor.Yellow;
  Console.WriteLine("B L A S T O F F ! ! !");
}
catch (OutOfMemoryException mex)
{
  Console.WriteLine("Not enough free memory to lauch shuttle.");
  Console.WriteLine("Please try again later.");
}
```

Hopefully, the real NASA application is a little more complicated than this, but you could imagine a situation where running out of available memory just after ignition could be somewhat problematic. It's better to know before you begin that

your countdown is likely to fail. Adding a MemoryFailPoint around the critical code will help with this:

```
try
{
  using (MemoryFailPoint mfp = new MemoryFailPoint(MB_MEMORY_REQUIRED))
  {
    …
  }
}
catch (InsufficientMemoryException mex)
{
  Console.WriteLine("Not enough free memory to lauch shuttle.");
  Console.WriteLine("Please try again later.");
}
```

When using MemoryFailPoint for more realistic purposes, you can make intelligent decisions about the execution path. For example, fast sorting algorithms can use more memory than slower but less resource hungry versions.

Secret Squirrel - System.Security.SecureString

What MSDN says:

> *Represents text that should be kept confidential. The text is encrypted for privacy when being used, and deleted from computer memory when no longer needed.*

This sounds like something useful to do, and indeed it is! To use a SecureString you need to append characters to it like so:

```
string password = "Password";

using (SecureString sPassword = new SecureString())
{
  foreach (char ch in password) sPassword.AppendChar(ch);
  // do something with the secure string
}
```

Clearly, this would be a tad pointless! The unsecure password is right there in clear text for all to see (and discover). To understand this, you need to understand show strings in the .NET Framework work.

Strings are imputable. That is, once you create a string in memory, it can't change – it can only be replaced. For example:

```
string somestring = “fred”;
// do something
somestring = “bob”;
```

Looking at this you might think that there is some chunk of memory somewhere that starts off as “fred” and later becomes “bob”. This is not the case. In actuality, both “fred” and “bob” will be in memory somewhere within the process space of the application. Every time you change a string, a new copy of the string is made and the old one is left for later cleanup by the garbage collector. A sniffer type application on a compromised machine could read through your unprotected memory quite easily.

So, when should you use SecureString? There are a few places in the framework that require a SecureString but the best time to use this is anywhere in your application that you prompt for something you don’t want to risk exposing to unauthorized users or processes.

SecureString should be used in conjunction with other secure mechanisms such as an encrypted configuration file or database record, SSL/TLS for web applications to network transport, etc.

References:

B# .NET Blog
http://bartdesmet.net/blogs/bart/archive/2006/03/31/3851.aspx

.NET Security Blog
http://blogs.msdn.com/shawnfa/archive/2004/05/27/143254.aspx

Peter Jones is a 25 year veteran (vintage?) programmer from New Zealand. He currently works for Intergen - a leading Microsoft Gold Partner. Peter has worked on and created systems for many industries such as Telecommunications, Manufacturing, Government, Education and more. He cut his programming teeth on dBase 3, Clipper 5 and C before moving to Windows development tools such as Visual Basic, Delphi and .NET. Currently he is working with content management systems (SharePoint and EPiServer) in Sydney, Australia. You can contact Peter and read his occasionally updated blog at ***http://jonesie.net.nz****.*

Creating Useful Installers with Custom Actions

by Christian Jacob

So you are almost done with implementing a ground breaking, state of the art, super duper plugin-enabled calculator application with syntax highlighting and now you are wondering how to get it deployed? In case you are already trying to put some arguments together to convince your boss for buying a suite to building Microsoft Installer packages worth several thousand dollars: **Stop! Don't do that**. Well, at least not if your application is not as complex as let's say your development environment.

Using Microsoft Visual Studio you should be able to create an MSI in literally no time by using the predefined Setup Project template and if you need to accomplish additional tasks that Windows Installer does not support out of the box, read on to find out how to extend your installer with own custom actions.

The Vision

A couple of years ago market researcher Evans Data noticed a massive drop of C++ developers and also forecasted a continuous decrease. Bjarne Stroustrup on the other hand said that the number of C++ developers is higher than ever (he's the inventor of C++ by the way). Whoever you trust, you are most probably reading this because you want to know how to use C# or VB.NET for writing custom actions in Visual Studio instead of using good old C++. So let's dive into that topic by creating a not so fictional case scenario: the first task you would want to accomplish is creating an eventlog source for your application (because you read that the installation of an application is the suggested moment for doing just that).

Note: We will do a lot more than that, so I have included a small sample solution for you to easily reproduce what I am talking about. You can download it from **http://devshaped.com/files/MyCalcSln.zip**.

The Solution

As I said earlier, you could use C++ to implement a native custom action. However, the easiest solution for getting this done using C# or VB.NET is deriving a class from the *Installer* Class and overriding the *Install* method. Then you would add your custom action to the *Custom Action Table* of the installer.

So let's start!

Create the Custom Action

Add an *Installer* by right clicking your project and select Add > New Item > Installer Class. Name it CreateEventSource.cs for now.

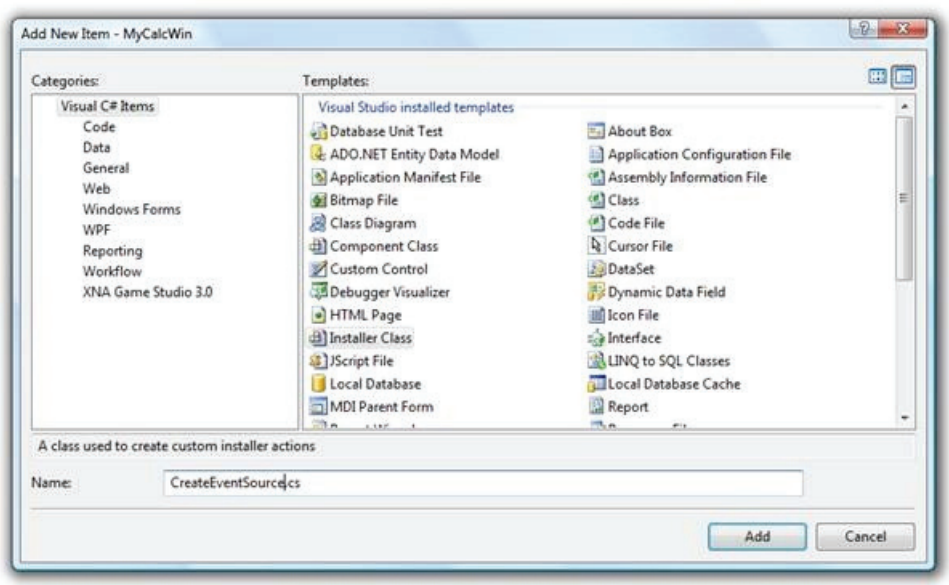


Figure 1- Adding an Installer Class

Next right click the CreateEventSource.cs file in the solution explorer and select View Code. Replace it with the following (you may want to adjust the namespace and the actual event source name):

```
namespace MyCalcWin.Custom_Actions
{
  using System;
  using System.ComponentModel;
  using System.Configuration.Install;
  using System.Diagnostics;

  [RunInstaller(true)]
  public partial class CreateEventSource : Installer
  {
    const string EventSourceName = "MyCalc";

    public CreateEventSource()
    {
      InitializeComponent();
    }

    public override void Install(System.Collections.IDictionary stateSaver)
    {
      try
      {
          base.Install(stateSaver);
          EventLog.CreateEventSource(EventSourceName, "Application");
          EventLog.WriteEntry(EventSourceName, "Eventsource created");
      }
      catch (Exception ex)
      {
        System.Windows.Forms.MessageBox.Show(
        string.Format("Error performing installer tasks: {0}", ex.Message));
      }
    }

    public override void Uninstall(System.Collections.IDictionary savedState)
    {
      try
      {
        base.Uninstall(savedState);
        EventLog.DeleteEventSource(EventSourceName);
      }
      catch (Exception ex)
      {
        System.Windows.Forms.MessageBox.Show(
        string.Format("Error permorming uninstaller task: {0}", ex.Message));
      }
    }
  }
}
```

As you can see, deriving from the *Installer* base class is not enough. You need to add the attribute *RunInstaller* to the class and initialize it with true as shown above. This basically marks the *CreateEventSource* class as an *Installer* class and makes it visible to reflection.

Create the setup project

Simply add a new Setup Project to your solution by right clicking your solution in the solution explorer and then Add > New Project > Other Project Types > Setup Project as shown in the following figure:

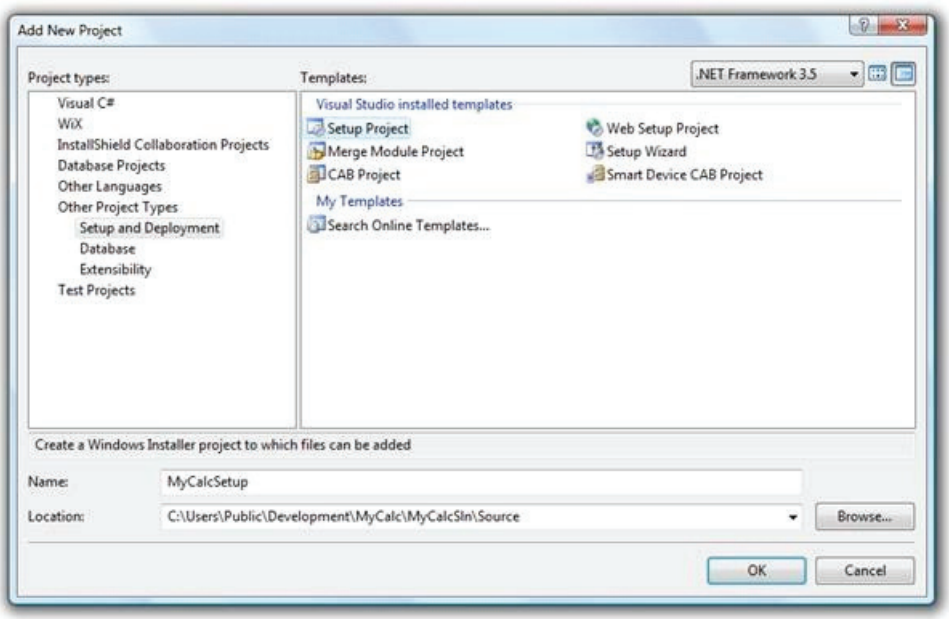


Figure 2 - Adding a Setup Project

To let the Setup Project know what to install, right click the project and select Add > Project Output… Make sure your application project is selected as Project and choose Primary Output. Click OK.

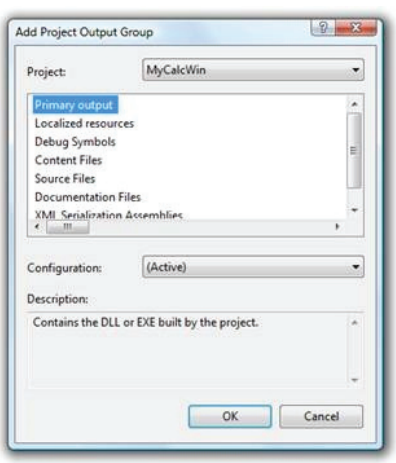


Figure 3 - Adding Project Output to Setup Project

With these simple steps you just created a setup that is able to install your application. Now you would make sure your custom actions are called and configure some simple settings. Right click the Setup Project in the solution explorer and select View > Custom Actions.

Now right click the Custom Actions node and select Add Custom Action. Look in File System on Target Machine and double click Application Folder. Select the Primary Output from your application project and click OK.

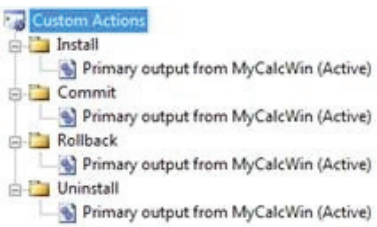


Figure 4 - Custom Action View

Easy so far! This will create the eventlog source for you by making use of the Installer Class you created earlier. If you would build your setup now and install it, the application will be defaulted to [ProgramFilesFolder][Manufacturer]\[ProductName]. Of course it is up to you to change the default target path. In my case, C:\Program Files\TOP TECHNOLOGIES CONSULTING GmbH\MyCalc is a little long. So I right click the Setup Project in the solution explorer and select View > File System and choose Application Folder. In the properties pane, I can configure the DefaultLocation to something like [ProgramFilesFolder][ProductName]. That's better.

Install your application

Now let's finally make a real job of it. Right click the Setup Project in the solution explorer and select Build. After being built successfully, right click the project again and select Install.

You should be familiar with the dialogs that follow. However, after your setup successfully installed your top notch calculator, have a look at the event viewer by clicking Start and entering eventvwr.

You will find the following entry in the application log:

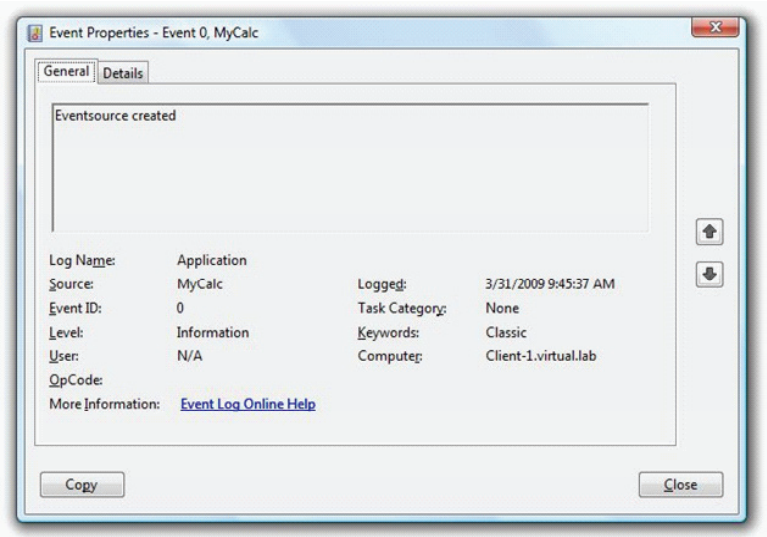


Figure 5 - Eventlog Entry

Observations

When using a fully featured environment for creating and using Custom Actions such as InstallShield from Acresso or Windows Installer Xml from Microsoft, you have absolute control over when and how your custom action is actually called. You can start it during the interview phase of your setup (aka UI sequence), during the immediate phase of the execution sequence when the actual installation script is generated or use it as a deferred custom action that is run in within the system context while the system changes are really applied.

Custom Actions implemented as Installer Classes are **only run deferred in the system context**. That means that your application is already successfully installed at that time. You have full access to all your files and also the configuration. You could implement a Custom Action for instance that alters configuration settings of your application based upon properties set by the user during the UI sequence (I will show you how later).

If you want to dive into the details of the Microsoft Windows Installer installation process, go ahead and read Alex Shevchuck's TechNet entry at **http://blogs.technet.com/alexshev/archive/2008/02/21/how-windows-installer-engine-installs-the-installation-package.aspx**. This is a great resource for a decent understanding about what's going on under the hood.

Potentialities (or: What the heck is CustomActionData?)

When looking at the choices in the View context menu of your Setup Project, you already noticed the following entries:

- File System
- Registry
- File Type
- User Interface
- Custom Actions
- Launch Conditions

What most certainly caught your attention is the User Interface entry. Yes! You can really add custom UI dialogs to a Visual Studio Setup Project. Although it is possible creating real custom dialogs, you can instantly start using own dialogs by choosing from the provided templates. Since we already agreed that we are talking about relatively simple applications here, in most cases that will get you started immediately (if you really really need more than that, look at **http://www.codeproject.com/KB/install/vsSetupCustomDialogs.aspx**).

Changing Application Settings During Installation

Let's have some fun with custom dialogs to do something really useful here you most probably won't instantly find by poking keywords into a search engine. Looking into the functional specification of your calculator you notice that one of your bosses (the one who usually has the most intriguing ideas of all) demanded that the calculator should welcome the user with his/her name. You heard him mumbling something about "Unique Selling Point" while he left the conference room…

So here is what you do:

Create a splash screen

Right click your calculator application and select Add > Windows Form. Name the form Welcome.cs and set its properties to the following values:

Property	Value
FormBorderStyle	None
ShowInTaskbar	False
TopMost	True
StartPosition	CenterScreen

Drag and drop three label controls into the form like this:

Figure 6 - Borderless splash screen

Create and bind an application setting to a label

Now configure the UserNameLabel label. In the property pane expand the ApplicationSettings node and click the ellipsis button (…) to the right of the Property Binding setting. Select the Text property, expand the drop down list and select New… Enter UserName as Name and Application as Scope (we'll assume that the user who installs the calculator won't let anyone else start it).

At this moment, Visual Studio automatically creates an app.config file for you that should basically look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
      <sectionGroup name="applicationSettings"
                    type="System.Configuration.ApplicationSettingsGroup,
          System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
          <section name="MyCalcWin.Properties.Settings"
              type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0,
              Culture=neutral, PublicKeyToken=b77a5c561934e089"
              requirePermission="false" />
      </sectionGroup>
  </configSections>
  <applicationSettings>
      <MyCalcWin.Properties.Settings>
          <setting name="UserName" serializeAs="String">
              <value />
          </setting>
      </MyCalcWin.Properties.Settings>
  </applicationSettings>
</configuration>
```

If you have a close look at the Welcome.Designer.cs file you will find the following line that does the binding for you:

```
this.UserNameLabel.Text = global::MyCalcWin.Properties.Settings.Default.UserName;
```

Let the splash screen appear at startup

To make the splash screen appear, adjust the Load method of your main form like this:

```
private void Form1_Load(object sender, EventArgs e)
{
  this.Hide();
  var splashScreen = new Welcome();
  splashScreen.Show();
  splashScreen.Update();
  System.Threading.Thread.Sleep(5000);
  splashScreen.Close();
  this.Visible = true;
}
```

Now let's finally wire things up with the setup project.

Create a custom dialog

Right click the Setup Project in the solution explorer and select View > User Interface. Right click the Start node in the Install tree and select Add dialog. Choose Textboxes (A) and click OK.

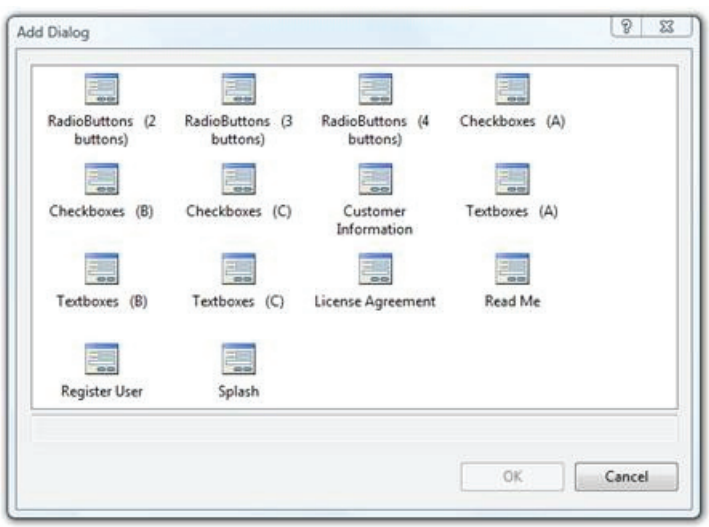


Figure 7 - Adding a Custom Dialog

Drag and drop the new dialog in front of the Confirm Installation node. Select it and change its properties to the following values:

Property	Value
BannerText	Enter your name
BodyText	MyCalc is the only calculator that will welcome you personally on each startup! Just tell it who you are.
Edit1Label	Your name:
Edit1Property	USERNAME
Edit2Visible	False
Edit3Visible	False
Edit4Visible	False

Now right click again the Setup Project in the solution explorer and select View > Custom Actions. Select Primary Output from MyCalcWin (Active) right under the Install node and change its CustomActionData property value to /userName="[USERNAME]".

Create a new Custom Action

Now comes the tricky part. We are talking here of at least a .NET 2.0 application. As we already saw, we are working with real type safe and supposedly easy to use settings utilizing the data bound Settings.settings file. Fortunately, .NET already generated a type safe wrapper for us. Unfortunately you cannot access the settings as easily as from within the application due to the fact that the installer simply does not have a mapper for it (and we won't supply it with one).

What you can do though is this:

Create another custom action as you already did before, call it RegisterUser and replace the generated code with the following:

```
namespace MyCalcWin.Custom_Actions

{
  using System.Collections;
  using System.ComponentModel;
  using System.Configuration;
  using System.Configuration.Install;
  using System.Xml;

  [RunInstaller(true)]
  public partial class RegisterUser : Installer
  {
    public RegisterUser()
    {
      InitializeComponent();
    }

    public override void Install(IDictionary stateSaver)

    {
      base.Install(stateSaver);
      // Get username from context
      var userName = this.Context.Parameters[“userName”];
      // Context contains assemblypath by default
      var assemblyPath = this.Context.Parameters[“assemblypath”];
      // Get UserName setting in app.config
      var config = ConfigurationManager.OpenExeConfiguration(assemblyPath);
      var sectionGroup = config.GetSectionGroup(“applicationSettings”);
      var section = sectionGroup.Sections[“MyCalcWin.Properties.Settings”]
                    as ClientSettingsSection;
      var settingElement = section.Settings.Get(“UserName”);
      // Create new value node
      var doc = new XmlDocument();
      var newValue = doc.CreateElement(“value”);
      newValue.InnerText = userName;
      // Set new UserName value
      settingElement.Value.ValueXml = newValue;
      // Save changes
      section.SectionInformation.ForceSave = true;
      config.Save(ConfigurationSaveMode.Modified);
    }
  }
}
```

As you can see, we can access the Custom Action Data we prepared earlier through the InstallContext Parameters collection. Also the InstallContext already contains a couple of default parameters such as the assemblypath (which we need to access the application’s configuration file).

What we are doing here is loading the .exe.config file of our application, digging through to the UserName setting, creating a new Xml node containing the username value and finally putting the value into the configuration before we are saving the changes. It is really as easy as that. Wasn’t that obvious?

Now compile and install your application.

The setup will ask you for your username:

Figure 8 - User registration dialog

When starting the calculator, the splash screen welcomes you:

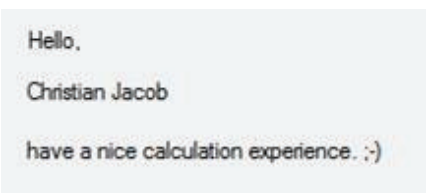


Figure 9 - Actual splash screen

Alternatives

Over and above the samples you saw so far there are several alternatives to using the Installer Class provided by Visual Studio. One is using native C++ (the languageWindows Installer is actually developed in, which is why it is the most powerful way of implementing custom actions). Explaining how that works is out of the scope of this article, but you can read about it for example on Steven Bone's Blog who published three absolutely awesome articles about it (**http://bonemanblog.blogspot.com/2005/10/custom-action-tutorial-part-i-custom.html**).

A native custom action that accesses the MSI log for instance basically looks like this:

```
extern "C" UINT __stdcall Install(MSIHANDLE hInstall)
{
  PMSIHANDLE hRecord = MsiCreateRecord(0);
  MsiRecordSetString(hRecord, 0, TEXT("Native C++ is better than Installer Class!"));
  MsiProcessMessage(hInstall, INSTALLMESSAGE(INSTALLMESSAGE_INFO), hRecord);
  return ERROR_SUCCESS;
}
```

Using DTF (Deployment Tools Foundation), which is part of the Windows Installer Xml Toolset, enables you to implement Custom Actions with managed code. However, you need to make sure .NET Framework is already installed on the computer you are running the setup on and also have a couple of prerequisites to fulfill.

Action	Installer Class	Native C++ DLL	WiX DTF
Run in UI sequence		X	X
Run immediate in execute sequence		X	X
Run deferred in execute sequence	X	X	X
Read MSI properties	X [1]	X	X
Write MSI properties		X	X
Debuggable	X	X	X [2]
Access MSI Logfile		X	X
Access Installstate log	X		
.NET Framework must be installed	X		X
Windows Installer Xml must be installed			X

Conclusion

In many cases, installation needs are straightforward enough that it is not worth the effort to implement anything more complex than the Custom Actions that run deferred in system context. Although you gain more control over the installation process when you choose to implement Windows Installer Xml based Custom Actions or native CA's, Installer class based Custom Actions can be a great way to extend your Visual Studio based Setup Project without the need to invest in additional installer solutions.

[1] Through stateSaver dictionary collection

[2] Simple unit testing possibility via wrapping the MSI session

Have a comment about this article? Post it at ***http://devshaped.com/2009/04/creating-useful-installers-with-custom-actions/****.*

Christian Jacob is an IT Consultant with TOP TECHNOLOGIES CONSULTING, a Microsoft Gold Partner with focus on secure infrastructures, development and management consulting. He primarily uses C# and VB.NET in nowadays development and has his roots in C++ based game development. He spends a lot of time in researching and creating innovative solutions around team development, Windows Installer, and TOP TECHNOLOGIES' main software products. He currently resides nearby Hamburg in North Germany.

Banking with XML

by Peter Jones

Remember XML? Remember those fun times you had creating XML data files, XSLs, and DTDs using Notepad? It seemed like such a great thing to do. There were grandiose plans to save the world with markup.

Today, XML is just another file format thanks to great tools and (reasonably well followed) standards. The tools in Visual Studio do such a great job of encapsulating XML that we often forget that it even exists under the covers.

And then along comes a scenario where you need to dig deep and reach for those reference manuals (or search engine). Recently I had just such a challenge.

The Mission

Create a site for Students and Law Professionals to register for seminars offered by a law college. Seminar attendees would pay for seminars using a credit card. Payments would be processed online by the college's bank.

The Solution

The web site was created using ASP.NET MVC, Visual Studio 2008 and SQL Server 2005. The college's bank provided various documents to describe their payment gateway interface which supports payment by the usual credit cards, plus credit transactions and other services. For this site we only required the payment service.

Principals

The .NET Framework is huge (**http://blogs.msdn.com/brada/archive/2008/03/17/number-of-types-in-the-net-framework.aspx**). It contains over 11,000 types (classes, structs, etc). You can bet your bottom dollar that there are several ways to do just about any task you choose. My overriding principal when creating any coded solution is to use the clearest and easiest to maintain code. Performance and elegance normally come second to this - mostly because of the types of applications I create but also because simple code is usually better code.

Getting Setup

For this exercise I have created a sample web service and client web site to test it. You can download the complete sample code from **http://devshaped.com/wp-content/uploads/2009/02/jonesiespaymentgateway.zip**. This sample includes a complete standalone project for processing payments with a test payment gateway. With a bit of work you should be able to easily adapt this for a real payment gateway of your choice.

In the remainder of this article I will be focusing on the **PaymentGatewayFormatter** class from the sample. A full explanation of the sample code is beyond the scope of this article but please feel free to drop me a line if you have any questions or suggestions.

The Problem

In these enlightened times, we tend to expect 'standard' web services to be made available for just about everything. Connecting to such a web service is trivial if you can use Visual Studio or the .NET SDK WSDL tool to generate the interface proxy for you. However, in this case, the bank provided only the following:

- A REST style web service sending and receiving plain old XML (POX) with no WSDL.
- A PDF document describing the service architecture, rules for usage, sample XML request/response messages, and DTD schemas.
- A PDF document describing the HTTP/XML and Java interfaces.

And that was it - nothing for .NET, no XSD, no WSDL and no SOAP. A call to the bank requesting a .NET interface resulted in a one word answer, which you can probably guess!

The Implementation

After some research and experimentation with Windows Communication Foundation (WCF) and Web Service Contract First (WSCF) and with limited time to create the interface, I opted for a hand coded implementation using low level framework support and HTTPWebRequest.

The Specification

The bank provided sample request and response messages as below:

Sample Request:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<TransactMessage>
  <MessageInfo>
    <messageID>8af793f9af34bea0cf40f5fb5c630c</messageID>
  </MessageInfo>
  <MerchantInfo>
    <merchantID>123456</merchantID>
    <password>password</password>
  </MerchantInfo>
  <RequestType>Payment</RequestType>
  <Payment>
    <TxnList count=”1”>
      <Txn ID=”1”>
        <txnType>0</txnType>
        <amount>1000</amount>
        <purchaseOrderNo>test</purchaseOrderNo>
        <CreditCardInfo>
          <cardNumber>4444333322221111</cardNumber>
          <expiryDate>08/12</expiryDate>
        </CreditCardInfo>
      </Txn>
    </TxnList>
  </Payment>
</TransactMessage>
```

Sample Response:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<TransactMessage>
  <MessageInfo>
    <messageID>b9cffb095b4f460fb0d60be85975b600</messageID>
  </MessageInfo>
  <RequestType>Payment</RequestType>
  <Status>
    <statusCode>000</statusCode>
    <statusDescription>Normal</statusDescription>
  </Status>
  <MerchantInfo>
```

```
    <merchantID>123456</merchantID>
  </MerchantInfo>
  <Payment>
    <TxnList count="1">
      <Txn ID="1">
        <txnType>0</txnType>
        <amount>1000</amount>
        <purchaseOrderNo>test</purchaseOrderNo>
        <approved>Yes</approved>
        <responseCode>00</responseCode>
        <responseText>Approved</responseText>
        <txnID>009844</txnID>
        <CreditCardInfo>
          <pan>444433...111</pan>
          <expiryDate>08/12</expiryDate>
          <cardType>6</cardType>
          <cardDescription>Visa</cardDescription>
        </CreditCardInfo>
      </Txn>
    </TxnList>
  </Payment>
</TransactMessage>
```

My first inclination was to create XSD files using these samples. Visual Studio made this easy to achieve. With the sample XML files open in Visual Studio I selected the Create Schema option from the XML menu. This produced an accurate XSD file. However, this was not much use by itself. I needed to create a C# class to include in my web site that would enable serialization of XML to and from the payment gateway.

The command line WSDL tool provided in the .NET Framework SDK does a great job of generating code for just this purpose. But in this case it was not so useful. Note that the request and response messages both have a root node of <TransactionMessage>. When I generated the C# classes from the generated XSD files, I had duplicate classes in the files for TransactionMessage, MessageInfo, Status etc. Fixing this would have required hand coding of the schema to include request and response messages, or merging of the generated classes. This felt like a lot of error-prone work.

XmlTextWriter

All I really wanted was a simple way to send a bunch of XML to a web site and read the response. The XML was not particularly complex or large so I opted to create the request using XmlTextWriter.

XmlTextWriter could not be any easier. I'll explain how to use it in the steps below.

1. Create a method to compose the required XML. I use a custom object, PaymentRequest to encapsulate the details of the request.

```
public string GetPaymentRequestXML(PaymentRequest request)
{

}
```

(All of the following code belongs inside this method).

2. Create a stream for the XmlTextWriter to write to.

```
MemoryStream ms = new MemoryStream();
```

3. Create an instance of an XmlTextWriter. Note the use of UTF8 encoding to match the sample XML file.

```
XmlTextWriter writer = new XmlTextWriter(ms, System.Text.Encoding.UTF8);
```

4. Use the XmlTextWriter to write elements and attributes to the stream. XmlTextWriter provides methods that produce validated elements only. For your example, we begin with the root node, TransactMessage:

```
writer.WriteStartElement("TransactMessage");
```

Every element that is started must be ended:

```
writer.WriteEndElement(); // TransactMessage
```

I find it less error prone to always create the start and end element method calls in pairs, and build the structure from the outside in - much like matching curly-braces in C#. Adding a comment to the WriteEndElement method call will also help you remember what you are ending. For example:

```
writer.WriteStartElement("MessageInfo");
writer.WriteElementString("messageID",
                          request.MessageID.ToString("N"));
writer.WriteEndElement(); // MessageInfo
```

This will produce the following XML in the output stream:

```
<MessageInfo>
<messageID>8af793f9af34bea0cf40f5fb5c630c</messageID>
</MessageInfo>
```

Use WriteElementString to create a complete element with a simple string as its inner xml value. To create an attribute on the current element simply follow the WriteStartElement call with WriteAttributeString.

```
writer.WriteStartElement("TxnList");
writer.WriteAttributeString("count", "1");
```

5. Flush the writer.

```
writer.Flush();
```

6. Convert the stream to a string and return it.

```
string xml = null;
using (StreamReader sr = new StreamReader(ms))
{

  ms.Seek(0, SeekOrigin.Begin);
  xml = sr.ReadToEnd();
}
return xml;
```

And that's about it. XmlTextWriter has many more useful methods to deal with namespaces and other standard XML features, but for this simple example these are not required.

XmlReader

Writing an XML file is very simple. Reading one, unfortunately, is not quite as straight forward. The main issue is that you cannot assume the order of elements will totally match your expectations. In our Bank of Jones sample it may be quite feasible to receive a response with the MessageInfo, Status and MerchantInfo elements in any order.

XmlReader allows you to read elements in an expected order, in much the same way as you would write them with XmlTextWriter. This may work for a while but when dealing with external systems there is no guarantee that it will work forever. In fact, when dealing with banks, it's a very good idea to assume it will change at random and you will only know about this when your client code fails!

Fortunately, there is a way to deal with this. XmlReader allows you to processes the XML serially. An example is the best way to describe this. Here are the steps.

1. Create an empty method to process the response stream.

```
public PaymentResponse ReadPaymentResponse(Stream input)
{

}
```

2. Create an XmlReader instance, and somewhere to store the de-serialized response data. In my case this is a PaymentResponse object.

```
PaymentResponse rsp = new PaymentResponse();
XmlReader reader = XmlReader.Create(input);
rsp.Status = PaymentResponse.StatusEnum.Incomplete;
```

3. Read through the input stream using reader.Read() which will read through the input XML one 'item' at a time. The item could be a start element, an attribute, an end element, comment, white space and more. Each time you call .Read() it advanced the internal pointer of the reader to the next item and sets a number of properties that you can use to determine exactly what the reader is pointing at. This is a very efficient way of parsing large XML files/streams because it is a forward only, read only mechanism.

Below is the main code I use to process the input stream.

```
while (reader.Read())
{
  if (reader.NodeType == XmlNodeType.Element)
  {
    if (reader.Name == "TransactMessage")
    {
      while (reader.Read())
      {
        if (reader.NodeType == XmlNodeType.Element &&
        reader.Name == "MessageInfo")
        {
          readMessageInfo(rsp, reader);
        }
        else if (reader.NodeType == XmlNodeType.Element &&
                 reader.Name == "RequestType")
        {
          reader.Read();
          rsp.RequestType = reader.Value;
        }
        else if (reader.Name == "MerchantInfo")
        {
          readMerchantInfo(rsp, reader);
        }
        else if (reader.Name == "Status")
        {
          readStatus(rsp, reader);
        }
        else if (reader.Name == "Payment")
        {
```

```
          readPayment(rsp, reader);
        }
      }
    }
  }
}
```

This code uses a number of sub-processing methods that also advance the cursor through the input stream (I have excluded these from here for brevity but you can examine them in the sample source code provided). Breaking the processing method into smaller chunks like this makes the methods much clearer.

Below is the helper method to update the PaymentResponse with the CreditCardInfo node. The other processing methods follow the same pattern of looping until the node ends.

```
private static void readCardInfo(PaymentResponse rsp, XmlReader reader)
{
  while (reader.Read())
  {
    if (reader.NodeType == XmlNodeType.Element)
    {
      if (reader.Name == “pan”)
      {
        reader.Read();
        rsp.CardNumber = reader.Value;
      }
      else if (reader.Name == “expiryDate”)
      {
        reader.Read();
        rsp.Expiry = reader.Value;
      }
      else if (reader.Name == “cardType”)
      {
        reader.Read();
        rsp.CardType = reader.Value;
      }
      else if (reader.Name == “cardDescription”)
      {
        reader.Read();
        rsp.CardDescription = reader.Value;
      }
    }
    else if (reader.NodeType == XmlNodeType.EndElement &&
             reader.Name == “CreditCardInfo”)
    {
      break;
    }
  }
}
```

Alternatives

The .NET Framework provides several ways of reading and writing XML files. The 'legacy' method is to use XmlDocument. This still works well but tends to be rather inefficient, especially with large XML files. The construction of the XML with XmlDocument is simple but not as intuitive as with XmlTextWriter. Reading of data from XmlDocument generally requires navigating through the structure using XPath queries.

If the 3.5 version of the .NET Framework, the new XDocument provided with LINQ is an excellent option. Using XDocument I could construct the payment transaction thus:

```
XDocument doc = new XDocument(
  new XDeclaration("1.0", Encoding.UTF8.HeaderName, string.Empty),
    new XElement("TransactMessage",
      new XElement("MessageInfo",
        new XElement("messageID", request.MessageID.ToString("N"))
      ),
      new XElement("MerchantInfo",
        new XElement("merchantID", MerchantID),
        new XElement("password", Password)
      ),
      new XElement("RequestType", PAYMENT_REQUEST_TYPE),
      new XElement("Payment",
        new XElement("TxnList",
          new XAttribute("count", "1"),
          new XElement("Txn",
            new XAttribute("ID", "1"),
            new XElement("txnType", "0"),
            new XElement("amount",
              (request.Amount * 100).ToString("##########")),
            new XElement("purchaseOrderNo", request.OrderNumber),
            new XElement("CreditCardInfo",
              new XElement("cardNumber", request.CardNumber),
              new XElement("expiryDate", request.Expiry),
              new XElement("cvv", request.CVV)
            )
          )
        )
      )
    )
  );
```

This syntax is certainly easy to construct and read but as you can imagine, a large deeply nested XML file can look messy and finding a mismatched parenthesis or missing comma is not fun.

Reading XML with LINQ bypasses the need to create cryptic XPath queries but it is still a query driven process. For example, to read the MessageInfo node would require the following code:

```
var messageInfo = (from mi in ele.Descendants("MessageInfo")
                   select new
                   {
                    messageID = mi.Element("messageID").Value
                   }).SingleOrDefault();
```

This code is clear enough and in some ways safer than my example, but it is not available for .NET 2.0. For my situation, XmlTextWriter and XmlReader provided a clear, simple, and efficient mechanism for dealing with XML data.

Have a comment about this article? Post it at ***http://devshaped.com/2009/03/banking-with-xml/****.*

Peter Jones is a 25 year veteran (vintage?) programmer from New Zealand. He currently works for Intergen - a leading Microsoft Gold Partner. Peter has worked on and created systems for many industries such as Telecommunications, Manufacturing, Government, Education and more. He cut his programming teeth on dBase 3, Clipper 5 and C before moving to Windows development tools such as Visual Basic, Delphi and .NET. Currently he is working with content management systems (SharePoint and EPiServer) in Sydney, Australia. You can contact Peter and read his occasionally updated blog at ***http://jonesie.net.nz****.*

Sending Email

by Derek Hatchard

Sending email from a .NET application is incredibly simple:

```
using System.Net.Mail;
…
SmtpClient _smtp = new SmtpClient(“smtp.test.com”);
_smtp.Send(“from@test.com”,
           “derek@devshaped.com”,
           “Subject”,
           “Body”);
```

With just these few lines of code, you are sending an email from from@test.com to derek@devshaped.com. Pretty easy stuff.

But what if you need something a little more complicated? The example above assumes the standard SMTP port for email with no authentication and an unencrypted connection. Fortunately these “complications” are quite easy to deal with in .NET.

Different Port

If you need to send email on a different port, simply add the following line:

```
_smtp.Port = 587;
```

SMTP Authentication

If you are not sending through a local SMTP server, your app will likely need to authenticate itself. If you need to pass credentials to the SMTP server, use the Credentials property on the SmtpClient class:

```
using System.Net;
…
_smtp.Credentials = new NetworkCredential("username", "pwd");
```

Encrypted Connection

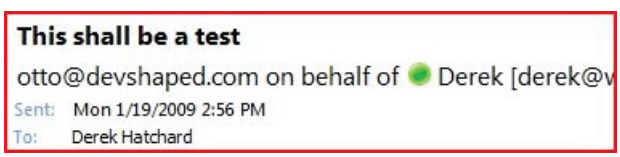
Since you are passing credentials to the server, you really should be using a secure connection (many mail providers require you to use a secure connection). In code, it's just one line:

```
_smtp.EnableSsl = true;
```

Sending Email On Behalf of Someone

A final practical consideration when sending email is the distinction between From and Sender. In a simple scenario, you only need a From address. But if you are writing code to send email, chances are good that there is something not-quite-so-simple happening.

Perhaps you have received an email with a header like this in Outlook:



This is an example of one email account sending an email on behalf of someone else. This is often used by automated systems and it is trivial to add to your application. To add a Sender you cannot pass values directly to an SmtpClient object; you must create a new instance of the MailMessage class, set the properties, and then pass the MailMessage object to the SmtpClient object:

```
MailMessage _msg = new MailMessage("from@test.com",
                                   "derek@devshaped.com",
                                   "Subject",
                                   "Body");
_msg.Sender = new MailAddress("sender@test.com");
_smtp.Send(_msg);
```

Keep in mind that your message is going through an SMTP server that might not let you set arbitrary From and Sender values.

Bringing it All Together

Here is what the final code looks like that incorporates an alternate port, credentials, encrypted transport, and a sender:

```
using System.Net;
using System.Net.Mail;
…
SmtpClient _smtp = new SmtpClient("smtp.test.com");
_smtp.Port = 587;
_smtp.Credentials = new NetworkCredential("username", "pwd");
_smtp.EnableSsl = true;

MailMessage _msg = new MailMessage("from@test.com",
                                   "derek@devshaped.com",
                                   "Subject",
                                   "Body");
_msg.Sender = new MailAddress("sender@test.com");
_smtp.Send(_msg);
```

Have a comment about this article? Post it at ***http://devshaped.com/2009/01/practical-programming-sending-email/****.*

*Derek Hatchard is the founder of Crowd Space (****http://crowdspace.net****). You can find him online at* ***http://derekhat.com****,* ***http://ardentdev.com****, and* ***http://twitter.com/derekhat****.*