UK MSDN Flash eBook: Best Technical Articles #2

January 2009 to December 2009

Eric Nelson, Technical Editor, Microsoft UK

16th December 2009

The UK MSDN Flash developer newsletter contains great short technical articles written by UK developers both inside Microsoft and in the broader developer community. This eBook pulls together these great articles in one place. There are thirteen articles in this second edition covering Python, Inversion of Control, Behavior Driven Development, Silverlight and more.

msdn
Microsoft® Developer Network

TABLE OF CONTENTS

INTRODUCTION

FROM THE EDITOR

Hello all,

Every two weeks we send an email out to tens of thousands of UK based developers. This email is called the MSDN Flash. The Flash contains many useful sections including a 400 to 500 word technical article on a developer related topic either written by a member of the Microsoft UK technical team or a member of the broader UK developer community. We have had some great articles over the years and this eBook includes the best articles over the period January 2009 to December 2009. It covers a variety of complex topics which have been condensed into short articles which take only a few minutes to read. You may also want to check out the first edition of the eBook which covered January 2008 to January 2009.

Please give us your feedback (30 seconds) once you have had a chance to look at the articles. Thank you and happy reading.

Eric Nelson
Developer Evangelist, Microsoft UK
Email: ericn.nelson@microsoft.com
Blog: http://geekswithblogs.net/iupdateable
Twitter: http://twitter.com/ericnel
Team twitter: http://twitter.com/ukmsdn

MSDN FLASH PODCAST

We also have a companion audio podcast to the Flash Newsletter which contains interviews with the authors of the technical articles. Many of the authors in this eBook have appeared in the podcast.

- Listen/download individual podcasts on channel 9
- Subscribe in FeedDemon or similar
- Subscribe in ITunes

WOULD YOU LIKE TO BECOME AN AUTHOR?

Developers value the sharing of best practices, knowledge and experiences. The MSDN Flash is a great way for you to do just that with thousands of other UK based developers.

The topics best suited to the MSDN Flash are topics **you** think a **lot** of UK based developers using Microsoft technology would like to know something **more about**. This makes for a pretty broad potential set of topics! Maybe you have something to share around Sharepoint development or Test Driven Development or perhaps you have war stories around configuring WCF for interop or using Object Relational Mappers to increase productivity. All are great candidates for becoming articles in the Flash.

Please email us with your proposed article(s) and if possible a “sample of your work” such as a link to your blog.

Email us at: UKMSDN@microsoft.com.

SUBSCRIBE TO THE UK MSDN FLASH

The MSDN Flash contains a lot more than the technical articles. It is designed to help you keep track of all the latest news on developer tools and technologies which are relevant to you. This fortnightly newsletter pulls together the most recent technical news in a concise and easy to read format so you can get all the latest information within a few minutes.

The UK MSDN Flash newsletter includes:

- **UK events and training conferences:** find local events, community meetings, training courses and more
- **Feature articles:** these articles cover the latest developer topics, real technical solutions and Microsoft insights
- **Fresh news, announcements and updates:** get the latest Microsoft updates, product announcements and product beta news
- **Recommendations and insights:** key highlights, resources, tips and offers available for developers

New to MSDN? Want to know more?

The Microsoft Developer Network (MSDN) is a set of online and offline services designed to help developers write applications for the Windows platform. To find out more about UK resources and programmes available from Microsoft, visit the UK Microsoft MSDN website. To stay connected with the UK developer community visit UK Channel 9 to find screencasts, podcasts and videos from Microsoft UK along with links to UK developer blogs, events and communities.

VISUAL STUDIO AND .NET FRAMEWORK

MEMORY MAPPED FILES WITH .NET FRAMEWORK 4.0

.NET 4 is on its way and there are many new features being blogged about, such as Parallel Computing and Dynamic Languages but the new feature that caught my eye was Memory Mapped Files. The idea of mapping memory to a physical file on disk has been around for a very long time on the Windows platform but this capability has not been available natively in managed code before and instead needed to be accessed via P/Invoke.

You might be wondering why you would wish to map memory on to a file, and why after numerous versions of the .NET Framework we need this feature now. To answer the first question it is worth pointing out that a memory mapped file is one of the most efficient ways for multiple processes on a single machine to talk to each other. What is more, most Windows IPC based communication mechanisms, such as COM, DDE, Sockets, Pipes and even Windows Messages themselves all sit on top of Memory Mapped Files. Another reason for using Memory Mapped Files is for processing extremely large files, as they enable random and concurrent access to a file without the need for seeking.

The large file processing is lovely and all that but I want to talk about inter-process communication on a single machine for a moment. There are many available options. You could go old school by using classic .NET Remoting for cross AppDomain communication by using transparent proxies, MarshalByRef and ContextBoundObject classes. You could code this using WCF and Named-Pipes; as well as all the available cross-machine techniques such as streams and sockets, or Web services by using one the various flavours of WCF Bindings or not forgetting plain old ASMX.

Given this wealth of options, when should you use Memory Mapped Files (MMF) over, say, WCF? For me, the answer is simple: it depends. As always with new technologies your millage may vary, but I think as a general rule, for IPC on a single machine MMF is the way to go; if only for the relatively simple programming model (see my blog post for a simple code example) and performance characteristics. Sometimes, when you need to send 4 bytes, you should just send 4 bytes; not a Soap envelope, encoded with a myriad of WS-* standards and human readable mark-up running into hundreds and hundreds of bytes – just to send your 4 byte message.

Memory mapped files will be my first choice for the scenarios that were previously reserved for *WCF and Named Pipes*, well, they will be as soon as .NET 4 ships. However, if you even suspect that at some point you might need your communication to spread across machines, then you are better off starting with WCF and named-pipes, because supporting cross machine communication in the future should simply be about making changes to configuration not recoding the communication layer.

That brings us back to the second question of: why now, why in .NET 4 and not some previous release? For that, we can only guess; but I am glad they finally did! You can find out more in the preliminary MSDN documentation.

PAUL JACKSON
Blog: http://compilewith.net
LinkedIn: http://www.linkedin.com/in/paulja

Podcast Available

Example code:

```
using System;
using System.IO;
using System.IO.MemoryMappedFiles;
using System.Text;

namespace PaulJ.MemoryMappedFiles
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create mapped file
            var mappedFile = MemoryMappedFile.CreateFromFile(
                new FileStream(@"c:\users\paul\desktop\memory.mp", FileMode.Create),
                "PJMemoryMap",
                1024 * 1024,
                MemoryMappedFileAccess.ReadWrite);

            // Write some data
            string value = "Hello World";
            var mapView = mappedFile.CreateViewAccessor();
            mapView.WriteArray<byte>(0, Encoding.ASCII.GetBytes(value), 0, value.Length);

            // Read some data
            var buffer = new byte[value.Length];
            mapView.ReadArray<byte>(0, buffer, 0, buffer.Length);
            var readValue = Encoding.ASCII.GetString(buffer);

            // Output
            Console.WriteLine("Data: {0}", readValue);
            Console.ReadLine();
        }
    }
}
```

LANGUAGES

WHY IRONPYTHON?

One of the new features in .NET 4.0 is the dynamic keyword, building on the Dynamic Language Runtime. A major reason for dynamic is to enable easier interaction with dynamic languages like IronPython. But if you're a dyed in the wool C# or VB.NET programmer why should you be interested in IronPython?

Much of the discussion here applies to other dynamic languages, including IronRuby, but Python is my particular area of expertise.

IronPython is a .NET implementation of the popular open source programming language Python. Python is an expressive language that is easy to learn and supports several different programming styles; interactive, scripting, procedural, functional, object oriented and metaprogramming. But what can you do with IronPython that isn't already easy with your existing tools?

The first entry in the list of programming styles is 'interactive'. The IronPython distribution includes ipy.exe, the executable for running scripts or programs that also doubles as an interactive interpreter. When you run ipy.exe you can enter Python code that is evaluated immediately and the result returned. It is a powerful tool for exploring assemblies and learning how to use new frameworks and classes by working with live objects.

The second reason to use IronPython is also the second programming style in the list; scripting. Python makes an excellent tool for generating XML from templates, automating build tasks and a host of other everyday operations. Because scripts can be executed without compilation experimentation is simple and fast. Python often creeps into businesses as a scripting language, but beware it spreads!

One of the big use cases for IronPython is for embedding in applications. Potential uses include user scripting, adding a live console for debugging, creating domain specific languages where rules can be added or modified at runtime or even building hybrid applications using several languages. Python has several features, like the ability to customise attribute access, that make it particularly suited to the creation of lightweight DSLs. IronPython has been designed with these uses in mind and has a straightforward hosting API.

There are many areas where dynamic languages are fundamentally different from statically typed languages, a topic that rouses strong opinions. Here are a few features of IronPython that make it easy to develop with:

. No type declarations
. First class and higher order functions
. No need for generics, flexible container types instead
. Protocols and duck-typing instead of compiler enforced interfaces
. First class types and namespaces that can be modified at runtime
. Easier to test than statically typed languages
. Easy introspection (reflection without the pain)
. Problems like covariance, contravariance and casting just disappear

The best way to learn how to get the best from IronPython is my book IronPython in Action. I've also written a series of articles aimed at .NET developers to help get you started including Introduction to IronPython, Python for .NET Programmers and Tools and IDEs for IronPython.

Happy experimenting!

Michael Foord
http://www.voidspace.org.uk/

TOOLS AND DEVELOPMENT PRACTICES

WHY DO I NEED AN INVERSION OF CONTROL CONTAINER?

You may have come across the term 'Inversion of Control Container' (IoC container), it's become a hot topic in discussions on .NET application architecture. The term 'IoC container' sounds quite scary, but the basic concept is simple. It is all about building component oriented software; software that can be re-wir..ed simply with configuration.

Before I introduce IoC containers, there's an essential building block we have to understand: Dependency Injection. It's another scary sounding concept, but it really is very straightforward. First an example using a class ***Reporter*** that gets reports and then sends them:

```
public class Reporter
{
   IReportBuilder _reportBuilder;
   IReportSender _reportSender;

   public Reporter(IReportBuilder reportBuilder, IReportSender reportSender)
   {
      _reportBuilder = reportBuilder;
      _ reportSender = reportSender;
   }

   public void SendReport()
   {
      Report report = _reportBuilder.CreateReport();
      _reportSender.Send(report);
   }
}
```

Rather than creating instances of classes directly inside ***Reporter*** using the 'new' operator, we are injecting our dependencies by passing anything that implements ***IReportBuilder*** and ***IReportSender*** to ***Reporter's*** constructor. Our software can decide what these classes will be after compilation. This is where the IoC container comes in. An IoC container is a very smart factory that knows how to create your class and its dependencies. We simply register our components and the interfaces they implement with the container. For example, stating that when we ask for an ***IReportSender*** we should be given an ***EmailReportSender***. Remember components are simply classes written using the Dependency Injection pattern show above. The IoC container uses reflection to examine the signature of ***Reporter's*** constructor and then looks up the components it needs in its configuration.

Now when we need a reporter we can simply get it from the container by writing:

```
Reporter reporter = container.Resolve<Reporter>();
```

The container will automatically resolve Reporter's dependencies for us, supplying the implementation of ***IReportBuilder*** and ***IReportSender*** that we specified in the configuration. It also supplies any dependencies that the actual ***IReportBuilder*** or ***IReportSender*** implementation might have; it wires up our entire object graph for us.

Now, what if I want to use ***SmsReportSender*** rather than my existing ***EmailReportSender*** throughout my application? I only have to change the one configuration entry, rather than searching through my entire application looking for every place where a new instance of ***EmailReportSender*** is instantiated.

You could write your own IoC container but it makes more sense to use one of the existing IoC containers for .NET, the best known is Windsor from the Castle Project.

Dependency Injection and IoC containers are based on simple concepts, but can yield a wide range of optimisations. *Decoupling* our software in this way means that we can write applications as a collection of components which can be tested and maintained independently. This is especially optimal with large enterprise systems. The ability to swap components in and out of an application without re-compilation is also extremely powerful.

I hope I've been able to spike your interest with this short article. I have posted a list of further reading on my blog if you would like to find our more.

Mike Hadlow
http://mikehadlow.blogspot.com/

TECHNICAL DEBT

So what is technical debt? Well, imagine it's Friday Afternoon, and it's Check in Time. You can't go home until you've checked in your code, and you can't check it in until the last test passes. To get that test to pass you know you should re-architect the object hierarchy slightly, but you know that's going to take a long time and there's a cold beer waiting for you at home. So what do you do? You apply a quick fix, knowing you're going to come back on Monday morning and fix it. We've all been there right? But how many of us remember to make that fix on the Monday morning? And that, dear reader, is technical debt. A little bit of code, just sitting there in the code base, gathering interest in the form of every subsequent piece of code that is build on top of it, or with which it has a dependency, just waiting for the day that debt has to be paid off.

Of course, technical debt is not always a bad thing. A little technical debt can speed the development process. Much like a business might go into financial debt to enable it to purchase the raw materials in order to manufacture the widgets for that big order they've just secured. In the same way, a development team might incur a little technical debt in order to meet an important milestone. Technical debt, like financial debt, only becomes a problem when it is left to get out of control and gather crippling interest.

There are a number of programmer habits that can contribute to technical debt; too many to list here. We'll look at two such habits. The first of these we'll call "Ignoring the Obvious". This anti-pattern occurs when a programmer can see that the underlying structure needs to be re-architected but carries on building on top of a "foundation of sand" either because of a perceived lack of time, or because he feels that it is not his responsibility to fix it.

The second such anti-pattern is the "Over-engineered Solution". In this anti-pattern a developer is so enamoured with the latest book he's read on code architecture that he is determined to use all the examples in his solution. This leads to a highly over-engineered solution, one where any maintenance developer is going to have to devote a lot of time to understanding what is happening instead of adding features to that code base.

Now that we know what technical debt is, how do we go about paying it off? In a word, refactoring. Refactoring is the process by which we fix the underlying coding issue(s) and pay off our debt. Fortunately there are tools on the market to help us do just that. CodeRush Xpress is one such tool and has the advantage of being free.

A good refactoring tool will enable you to navigate your code base and fix any underlying debt causing issues by highlighting the offending code and selecting the best solution from a choice of available refactorings. The best tools will even analyse your code for you and point out issues that perhaps you didn't even realise you had.

Gary Short
Read Gary's blog
Evangelist, Developer Express

Podcast Available

NEURAL NETWORKS

One idea that had always interested me was neural networks. Neural networks were inspired by our current understanding of how our brain works. Neuroscience suggests our brain is made up of a vast number of interconnected switches called neurons that fire electrical pulses to each other. Neural networks attempt to emulate this process to produce a more flexible program.

Neural Networks excel at problems such as recognising patterns and are commonly used in biometrics, games and financial analysis.

Many AI concepts contain tricky maths and neural networks are no exception. The basics though are actually pretty simple and to avoid the maths utilize an existing open source library such as Neurondotnet.

Imagine we want to create a neural network that will emulate a logical AND gate. The AND gate will have two inputs that can be either on (represented by 1) or off (represented by 0). Only if both inputs are set to 1 will the AND gate output a value of 1.

We will create a type of switch (a neuron) that will only send an output signal when the cumulative input reaches a certain threshold value. In our example we don't want any output signals unless both input signals are 1 so we will set this threshold to 2. Now we have set the threshold unless both inputs are 1 then our Neuron will output 0. We have created our AND gate.

Of course this isn't that useful and there are easier ways to perform this task. The real power of neural networks comes from their ability to learn and cope with complex data.

Let's imagine we don't know what value to set the threshold of our Neuron to. One way of finding out the threshold value is to supply our network with some training data and then compare the expected output with what we actually got to readjust the threshold values. Our training data will consist of pairs of inputs and the expected output for these values.

The training process is as follows:

1. Set neuron thresholds to random value
2. Pass training data into the neuron
3. Compare actual output from neuron against the expected output for the neuron and save this difference
4. Pass difference back through the network and use it to adjust the threshold values
5. Repeat the process many times until the difference between the actual and expected output is so small it is not really worth worrying about
6. The neural network is then considered trained and ready for use!

Real life neural networks can be much more complex than this and there are many different types.

If you want to know more I would recommend Jeff Heaton's book "Introduction to Neural Networks for C#" and John Wakefield's blog.

Happy reading!

In his free time Alex is writing "Introducing .net 4 with Visual Studio 2010" for Apress and runs the Surrey .NET user group DevEvening.co.uk.

Alex Mackey
Read Alex's blog

GETTING STARTED WITH THE MANAGED EXTENSIBILITY FRAMEWORK

In this article, I will provide a basic overview of the Managed Extensibility Framework (MEF), and detail a practical sample of providing extensibility in your application.

But what is it? And why is using it a good idea?

According to its project web site, MEF is "a new library in .Net that enables greater reuse of applications and components." MEF provides an engine that will dynamically extend your .NET application with plug-ins at run-time. You define the requirements to extend your application by decorating the classes in your core application with MEF's attributes. Plug-ins can then be created as classes that will satisfy these requirements, again using MEF's attributes to indicate what requirements the plug-in class will satisfy.

Currently in open development, the October 2009 release (Preview 8) is licensed under the MS-PL – which means you can use it right away for real projects, and MEF will also be included as part of .Net 4 in both regular CLR and Silverlight incarnations. There is already a strong community grown up around MEF, including a Contrib project and plenty of articles on the web – but getting started with MEF can be a daunting prospect.

Glenn Block put it very succinctly: "In MEF **everything** is an extension, everything is extensible (extensions themselves included)".

Let's look at a simple sample (download the code) - a Login processing module which needs to allow extensions to the processing of a login request with custom code, without re-writing the login code and ***without*** the login module referencing the extensions.

To implement the extensibility for the login module, we define an interface that describes the functionality that a Login Extension should provide (**ILoginExtension**), an "extension point" in login code at the point where logically the checks should occur (in the sample this is in the **LoginHelper** class), and finally a **LoginExtensions** property (again in the **LoginHelper**) that will contain the extensions that MEF provides.

What makes MEF so easy is that the interface (**ILoginExtension**) exposed by the requirement is the only piece of shared information. MEF scans all assemblies in your application (and / or specific directories as you choose) for plug-ins that will satisfy the requirement and calls the setter for the **LoginExtensions** property, "injecting" the list of extensions so that the code at the extension point merely has to iterate the list and call the implementation method on each extension.

This magic happens when, in the **LoginHelper's** constructor, we call MEF's providers to generate a catalog of all extensions (classes with an **Export** attribute) and requirements (classes with properties decorated with an Import attribute). Better still, this catalog could watch the file system for new plug-ins being added on the fly.

This catalog is then passed to the composition engine (an instance of the MEF **CompositionContainer** class) and the container statisfies all the requirements (**Imports**) found with the matching extensions (Exports).

Even for this simple case, the MEF provides an elegant and above all ***simple*** way of defining and implementing extensibility within your application. Use it once, and you'll be using it again and again to implement extension points in your code that will make it much more flexible.

More information & the sample can be found on my blog at http://bit.ly/msdn_mef_intro

Joel Hammond-Turner
Read Joel's blog at http://bit.ly/goldennuggets

IS BDD JUST TDD WITH A DIFFERENT NAME?

Test Driven Development is about design, not about testing first. We use unit tests and testing frameworks to drive out the design of our API and validate its usage. Think of TDD as an approach to defining the specification of our application.

However the true purpose of TDD is not always completely understood. The word *Test* or testing framework constructs such as *TestClass, TestMethod* or *Assert* detract from the true goal of TDD. In response to this, Dan North came up with the term Behavior Driven Development (BDD) and a framework (*JBehave*) that supported BDD.

Take a look at the following test:

```
public class TestAuthentication()
{ // Test code …

}
```

This is meant to test some sort of authentication. Unfortunately it is ambiguous. Under what conditions is this test being run? Is it with a valid user? What should it return? When should it fail? How is it called? Who's making the call? There is no context.

This is where BDD tries to help. In BDD, context is key. A test, which is referred to as a specification, should always identify clearly the context. A context is a scenario or set of circumstances that give rise to a certain action with a specific result. Think of it as a user story. We move away from defining how a specific class or method is implemented and more towards how aspects of the system behave under a specific scenario or context. Dan North defined a template of *Given, When, Then* to portray this.

However, this isn't always easy to express in terms of traditional unit testing frameworks. For instance:

```
public class given_valid_user_when_validating_user_it_should_return_user()
{
// Test code …

}
```

This is where BDD frameworks such as *JBehave*, *RSpec* (for Ruby) or the recent *MSpec* for .NET can help. *MSpec*, which makes use of lambda expressions, would convert C# code into nicely formatted specifications (check out this example).

By focusing on behavior, removing some of the technical words and introducing a common language that both developers and businesses can understand, we can produce specifications that represent more accurately the customer's needs. As an added bonus, we obtain not only a precise specification, but also a functional one that can be validated.

Hadi Hariri

http://hadihariri.com/

Podcast Available

Screenshot of a BDD Test

The test is written using MSpec in C#. The ***Establish*** is used to setup the actual scenario. The ***Because*** is where the actual action takes place. Finally the ***It*** sections are where the assertions happen, where we verify the behavior.

```
[Subject(typeof(LoginController))]
public class when_user_logs_in_with_valid_credentials
{
    Establish context = () =>
    {
        var authenticationServices = new Mock<IPasswordAuthenticationServices>();

        var userServices = new Mock<IUserServices>();

        var user = new Customer("username", "firstName", "lastName", "email@email.com", "");

        userServices.Setup(m => m.GetUserByUsername<User>("username")).Returns(user);

        loginController = new LoginController(authenticationServices.Object, requestContext.Object, userServices.Object);

    };

    Because of = () =>
    {

        redirectResult = loginController.Login("username", "password") as RedirectToRouteResult;
    };


    It should_redirect_to_homepage       = () =>
    {
        redirectResult.RouteName.ShouldEqual(RouteNames.CustomerHomePage);
    };

    It should_set_the_current_loggedin_user = () =>
    {
        requestContext.Verify(m => m.SetCurrentUser(Moq.It.IsAny<string>(), Moq.It.IsAny<string[]>()), Times.Once());
    };


    static Mock<IRequestContext> requestContext = new Mock<IRequestContext>();
    static RedirectToRouteResult redirectResult;
    static LoginController loginController;

}
```

Arrange

Act

Assert

Assert

Screenshot of the output

Output of the test run using TestDriven.NET. The text is generated using the class name and the ***It*** declarations to produce a nicely formatted output – underscores are removed.

```
Output
Show output from: Test
------ Test started: Assembly:                .Specification.dll ------

LoginController, when user logs in with valid credentials
» should redirect to homepage
» should set the current loggedin user
```

TEST DOUBLES, MOCKING AND STUBS

I'm fairly sure you will have been using Test Doubles yourself for longer than you think; you just may not have called it a Test Double. Have you ever written a skeleton class that implements an interface, just so it could be used in a call to another object? Did this class simply return some constant value from its methods? Perhaps you were writing a unit test and didn't want to have to setup a 'real' database just for your test. This is a Test Double.

A stub is a type of Test Double that, in the simplest terms, returns predetermined values from method calls. For example imagine you are testing a controller for a banking application. You want to ensure that when the controller method to display the balance in a different currency is called, the correct result is returned. You do not want to prove that the correct value is returned from your Account Service; merely that the controller calculates the result of a currency exchange correctly. Your code may look like this:

```
IAccountService mockAccountService = _mockRepository.Stub<IAccountService>();
SetupResult.For(mockAccountService.GetBalanceForAccount(12349876)).Return(250.00m);
_mockRepository.ReplayAll();

AccountController accountController = new AccountController(mockAccountService); Decimal balance =
accountController.BalanceInCurrency(12349876, Currency.USD);

//Result will be 500.00 as there are 2.00 USD to GBP, the Base Currency of the account
Assert.AreEqual(500.00m, balance);
```

There are other types of Test Doubles, the most commonly used form being Mocks. Mocks are used to test behaviour and care that they are called and called correctly. For example; did the account service above call the auditing service?

There are frameworks you can use to save yourself from the tedious task of creating Test Doubles. These frameworks include; Rhino Mocks, Typemock, NMock and Moq. Each framework has subtle differences in the way they work or are used, although they all meet the same goal. The types of Test Double each framework creates can differ, as does to some degree the terminology used. Moq for example makes no distinction between Stubs and Mocks.

In recent years high quality code has become a key goal. Tightly coupled objects are now widely recognised as a contributing factor in poor quality code. To reduce coupling developers use techniques like Dependency Inversion. When you are writing a test, you want to primarily concern yourself with the subject of your test. You don't want your test of a login controller, for example, to fail because one of the dependencies of the controller is broken. You want your test to fail only if the subject under test is broken. You can avoid such brittle tests by using Test Doubles in place of concrete interface implementations.

Hopefully this has given you a good introduction into the world of Test Doubles and Mocking.

Tom Quinn
Read Tom's blog

Podcast Available

WEB

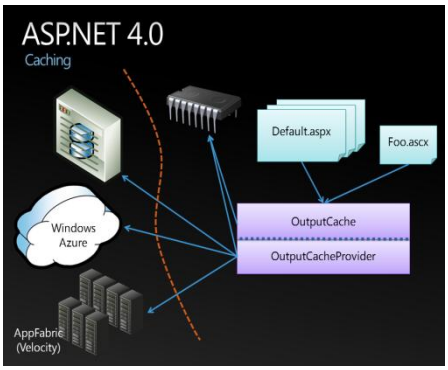
ASP.NET 4.0 WEB FORMS

Almost 10 years on and ASP.NET Web Forms is established as a mature web application framework, enjoying all the benefits of the .NET Framework, your choice of programming language and a rich ecosystem of 3rd party controls and frameworks (e.g. MonoRail and Spring.NET). The next releases of Visual Studio (2010) and ASP.NET (4.0) include a host of new features and enhancements. I'd like to tell you about a few of the changes targeted at Web Forms developers.

Some enhancements will be no surprise to many of you; they've either been on the wish list or represent a natural evolution of ASP.NET. Giving developers control over generated Client IDs for example is a welcome change. As is making it simple to disable ViewState when it's not required and enabling it only where needed. And the trend for cleaner generated markup continues. A good example in this release is the table-less rendering mode available for controls such as FormView.

In Visual Studio 2010, web developers at last get "code snippets" functionality for HTML, ASP.NET and Javascript as well as improved Javascript intellisense. The intellisense engine is faster and smarter to work better with 3rd party frameworks (the jQuery library will ship "in the box" with ASP.NET 4.0). For deployment, Visual Studio 2010 integrates with MSDeploy to enable the creation of web packages comprising the application, associated metadata, IIS settings, certificates and even SQL Server schema and data while web.config transforms allow easy deployment of unique settings for each configuration (e.g. different connection strings for debug, staging, release).

There are changes for caching, particularly with respect to extensibility; making it easy to plug-in different providers for output caching and create custom object cache implementations. SessionState gets a compression option for out of process scenarios where the network may be a bottleneck.



If you've used ASP.NET MVC you'll be familiar with the powerful new routing capabilities introduced in ASP.NET 3.5 SP1. ASP.NET 4.0 introduces those same routing capabilities to Web Forms applications making it possible to harness the same flexibility and friendly URL structure. New expressions have also been added for working with routes and route data declaratively.

Finally, ASP.NET AJAX takes a quantum leap, introducing simple yet powerful client-side templates and data binding allowing you to easily create data-driven UI on the client. Templates even support embedded code and expressions allowing you to do conditional rendering or modify the UI on the fly. The new DataView client control can bind to JavaScript objects or arrays and even has the smarts to call services to fetch its data. There's also support for change tracking and ADO.NET data services as well as an implementation of the observer pattern for JavaScript.

All this new functionality in ASP.NET AJAX has led to an increase in the size of the supporting library files. To counteract this, the library has been re-factored into more than a dozen separate components. You're now able to specify only the AJAX library files necessary to support the specific features you're using on the page. The new Script Loader takes care of working out which scripts it is you need.

If you want to know more, why not take a look at the whitepaper or the documentation on MSDN or download Visual Studio 2010 Beta 2 and have a play.

Mike Ormond Read Mike's blog

Podcast Available

CLIENT/OFFICE

DATA BINDING IN WPF

One of the most common complaints I hear about Windows Presentation Foundation (WPF) is that it's just about flashy visuals and doesn't really bring anything to the table that you couldn't do in Windows Forms (WinForms). Lots of WPF samples have been created which look great but exercise very few of the really useful features of WPF. To my mind, one of the best features of WPF is the ease with which you can bind to data of almost any type and optionally have two way data binding.

Where WPF outshines WinForms is through its ability to bind to things which are not a traditional data source. For example you may want to set the display size of an image to match the actual size of the image. This is easy to do in WPF:

<Image Source="{Binding MyImage}" Width="{Binding MyImage.Width}" Height="{Binding MyImage.Height}" />.

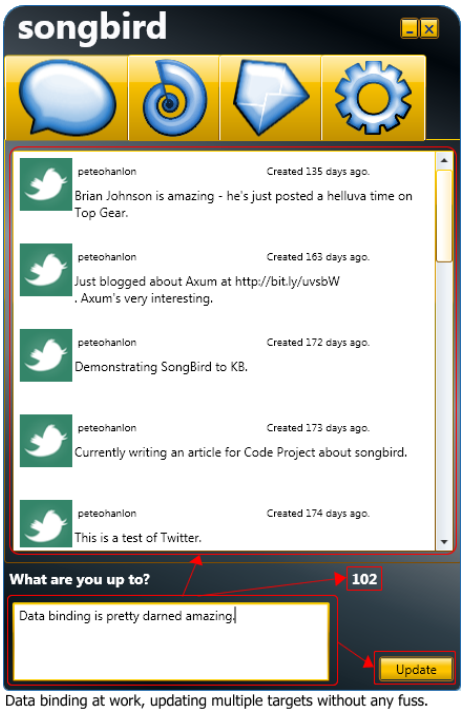
It is also perfectly possible to bind to "ancestors" of the current item. For instance you can set the current size of a grid in a user control to match the width of the container of the user control the grid is in, but have a different width for the control the grid is in – not something which can easily be achieved in WinForms.

Commonly we need to update the UI because of changes to your data, which in a WPF application we often surface as a data model. If your data model implements change notifications via INotifyPropertyChanged, then your UI can be updated in response to a change in the underlying data model. We've used this approach very successfully in order entry applications, where we can easily display a rapidly changing stock level and running totals. With this scenario, we don't need two way notifications. Hence we use a OneWay notification, where the target is updated whenever the data source is updated. For example:

```
<StackPanel Orientation="Horizontal">
<Label Text="Stock Level: " />
<TextBlock Text="{Binding StockLevel, Mode=OneWay}" />
<StackPanel>
```

Suppose that you want to keep track of the number of items in your order. Again, you can do this with binding. There is a fantastic collection called ObservableCollection which notifies you when items have been added or removed from it. With this, it's easy to display the total using a binding such as:

```
<StackPanel Orientation="Horizontal">
<Label Text="Total orders: " />
<TextBlock Text="{Binding Count, Mode=OneWay}" />
<StackPanel>
```



Data binding at work, updating multiple targets without any fuss.

Why am I so keen on binding in WPF? We use Test Driven Development to develop our applications which allows us to concentrate our efforts on getting the model and business layer right. Our UIs then bind to a data model (or ViewModel if you use the MVVM pattern) and all the hard work of displaying the data and handling updates is taken care of. This makes us more productive and allows us to turn out great applications in a lot less time – which is good news for our clients. For more detail on binding, please visit my blog.

Peter O'Hanlon Read Peter's blog

GENERATE OFFICE 2007 DOCUMENTS WITH THE HELP OF DOCUMENTREFLECTOR

Office 2007 is a fine example of how we have to maintain some degree of backward compatibility in our applications. Specifically, I am referring to Word, PowerPoint and Excel and their ability to manage a variety of subtly different binary formats that have evolved over many years. The advent of an XML-based file format is yet another feather in the cap for the Office 2007 file format developers. Including a text-based file format in Office 2007 was a huge undertaking that necessitated the creation of an entirely new object model that replaces the legacy COM offering found in Office 2003 and earlier.

The Open XML SDK provides us with a layered API for creating and working with Open XML documents. During the SDK's maturation, it went from a loosely-typed model through to a model boasting strong typing. Strong typing means we're baking quality into our applications at compile-time – there are no references to strings or XML node names in our code. Run-time errors due to spelling mistakes and assumptions become a thing of the past.

Out of the box, the Open XML SDK 2 includes three tools: DocumentReflector for code generation, OpenXMLClassExplorer to explore the Open XML markup (and the Ecma 376 specification) and OpenXMLDiff to graphically compare Open XML files. Personally, I really like the Document Reflector; it's a very powerful tool.

Those of you used to using Red Gate's .NET Reflector should be very familiar with the concept of reflection. The DocumentReflector is a small application that takes existing Word 2007, Excel 2007 or PowerPoint 2007 documents and generates the C# required to recreate the given document. It's a great way to learn the Open XML API – because XML documents are hierarchical, we can use a treeview for representation and navigation. The document reflector is able to show us the code required to create specific "parts" of an Open XML document.

If you have existing documents in Word or Excel, the DocumentReflector is a great way of embedding document creation in your application with very little development effort. It becomes the simple matter of cut'n'paste to include even the most complicated Word or Excel document creation. Of course, when you open a legacy (e.g. Word 2003) document in the Office 2007 counterpart, fidelity is maintained – old documents become addressable via C#.

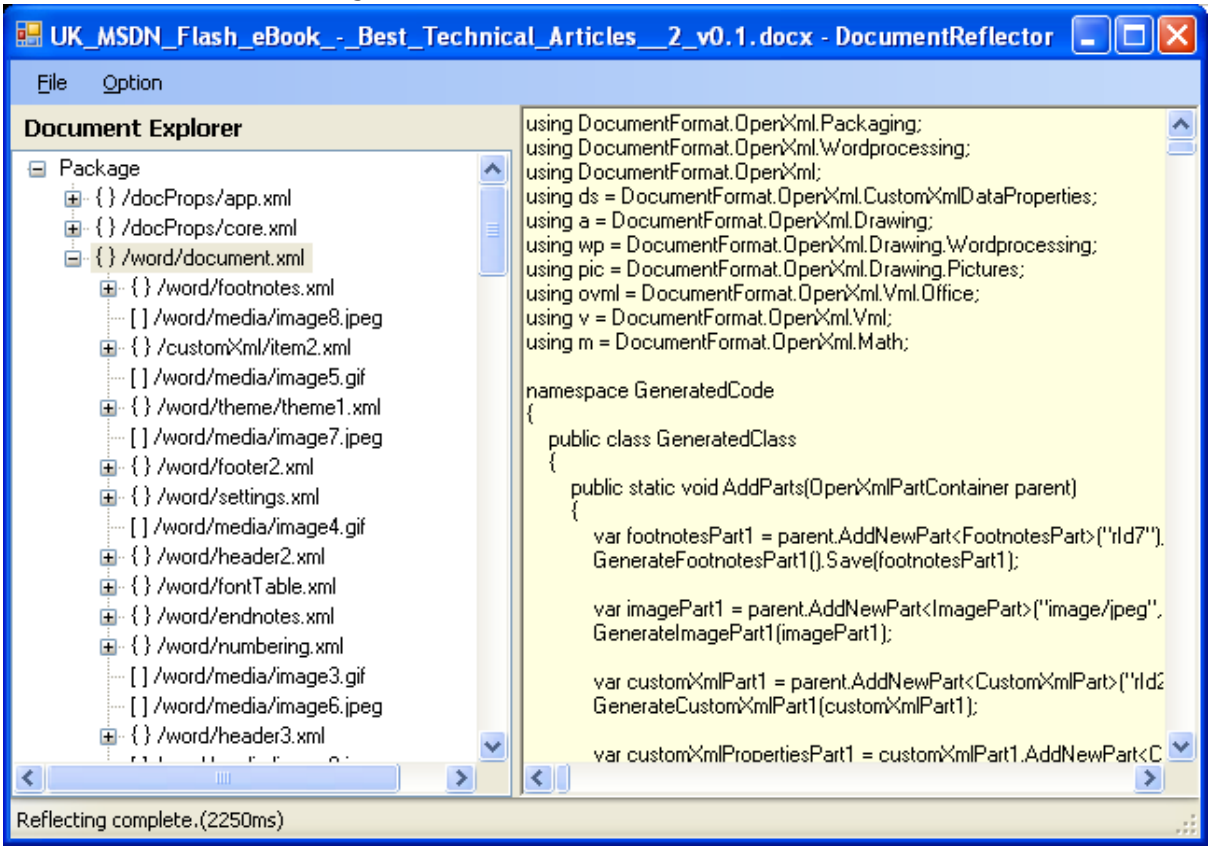
That all sounds great but can it get any better? Yes it can. You can do all of this without the need for an Office 2007 license on your deploy target. Now if that doesn't tempt you into looking at Open XML, I don't know what will!

Download the Open XML SDK and check out my list of Open XML resources and this great "getting started" post from Erika Ehrli.

Craig Murphy

Read Craig's blog
Follow Craig on Twitter

The DocumentReflector creating C# from a Word 2007 document:



Creating a Word 2007 document using C#

```
using (WordprocessingDocument package =
    WordprocessingDocument.Create(@"c:\z_WordML-2.docx",
        DocumentFormat.OpenXml.WordprocessingDocumentType.Document))
{
    package.AddMainDocumentPart();

    // Strong typing
    package.MainDocumentPart.Document =

        new Document(
            new Body(
                new Paragraph(
                    new Run(
                        new Text("MSDN Flash rocks!")))));

    package.MainDocumentPart.Document.Save();
```

MODEL-VIEW-VIEWMODEL GETS THE MOST OUT OF WPF AND SILVERLIGHT

Since Windows Presentation Foundation (WPF) was released over two years ago it has been changing the way we build User Interfaces thanks to its amazing data visualization capabilities and data-binding support. We've had data-binding before but the difference with WPF is that data-binding was baked into the platform from the beginning. For example, it's possible to implement complex functionality using data-binding alone - see how I created a ScrollViewer Thumbnail control for an example.

This power is enabling greater separation and is supporting an emergent pattern for developing WPF applications called Model-View-ViewModel or MVVM. Much like Model View Controller (MVC) and Model View Presenter (MVP), MVVM aims to:

- **Increase Separation** - Breaking code into multiple classes improves the readability and maintainability of a system as per concepts such as the Single Responsibility Principle.

- **Increase Testability** – Some presentation frameworks (e.g. ASP.NET WebForms) are difficult to unit test as the View is mingled with the code behind and there are many dependencies to be met to allow testing, increasing "test friction".

In MVP we extract much of the logic from the View and place it in a new class, the Presenter, which can be tested in isolation. However, the Presenter must be able to communicate with the View and we typically do so through an interface that can be simulated or 'mocked' during testing.

In MVVM we replace the Presenter with a class called the ViewModel that behaves in a similar manner but with one key difference: rather than communicate with the View directly, the ViewModel exposes properties that the View can data-bind to. This reduces test friction even further as the ViewModel has no reference to the View that needs to be mocked. A unit test can simply drive the ViewModel and test that its properties behave as desired.

Because of this, the use of MVVM can often eradicate the need for code in the View's code-behind file, which leads to the other great benefit of this pattern:

- **Designer / Developer Workflow:** WPF allows for an entirely customisable look and feel to be applied to an application. To do this well requires the skills of a Designer working alongside the Developer. In many User Interface frameworks these two roles would regularly need to work on the same assets - but with MVVM we create a physical separation that allows the two roles to work in harmony. The developer creates and agrees a ViewModel with the designer whilst the designer can work on the XAML that binds to the ViewModel.

Even in the absence of a designer, MVVM is still a hugely beneficial pattern that will drive your WPF development to the next level. And with the introduction of Silverlight 2 and its similar data-binding capabilities MVVM is finding favour with a new group of developers. Find out more about MVVM in this article: WPF Apps With The Model-View-ViewModel Design Pattern.

Josh Twist
Application Development Consultant

CONSUMING REAL-TIME DATA IN A SILVERLIGHT RIA

The term "real-time web" is in everybody's inbox at the moment. I believe that the "real-time web" is not just the immediate availability of data but also being supplied with the data in real-time. In this short article, I'd like to share some of the things to consider when consuming real-time data using an RIA technology, with some examples specific to Silverlight.

Real-time data can be consumed in a number of ways but there are a few techniques that are better suited to an RIA. A publisher/subscriber hub is one way of subscribing to data but this technique requires the subscriber to be running a web server which isn't possible from Silverlight or any other RIA technology. Continuously polling a resource is the most common way to check if the data you are interested in is available or has changed. This approach is known as client pull and used by WCF Duplex Services and might be fine if the data doesn't change frequently and you don't mind wasting the occasional server request when there's no new data available. However, if your RIA really needs data in real-time then I believe that HTTP server push is the best solution.

This HTTP server push technique can be achieved in Silverlight using the HttpWebRequest class but you need to be aware that there are URL access restrictions which you must obey, you can't set the Credentials property (since the browser should supply these), and you should set AllowReadStreamBuffering property to false (to ensure the response callback triggers).

```
public void Connect()
{
  HttpWebRequest request = (HttpWebRequest)WebRequest.Create(new Uri(URL));
  request.AllowReadStreamBuffering = false;
  request.BeginGetResponse(ConnectionResponseCallback, request);
}
// callback method continuously reads data.
private void ConnectionResponseCallback(IAsyncResult asynchronousResult)
{
  HttpWebRequest request = (HttpWebRequest)asynchronousResult.AsyncState;
  using (HttpWebResponse response =
(HttpWebResponse)request.EndGetResponse(asynchronousResult))
  {
    byte[] buffer = new byte[65536];
    using (Stream stream = response.GetResponseStream())
    {
      while (connected)
      {
        read = stream.Read(buffer, 0, buffer.Length);
        ParseResponse(buffer);
      }
      // call request.Abort or the the thread will
      // block at the end of the using block.
      request.Abort();
    }
  }
```

```
}
```

You should try and keep your RIA's resource usage as low as possible or the browser may stop responding. Consider the volume of data being sent by the server to the client and throttle the data updates to only send enough to ensure the data is fresh and the application useful. For example, watching the Twitter public timeline in real-time would be impressive but is incomprehensible to a user.

In Silverlight you need to remember that UI updates and web requests are using browser resources and competing for the UI thread. In Silverlight the System.Threading namespace is available so delegate to a non-UI Thread as soon as possible.

If you have the option of defining the format of the data your RIA is consuming then you should keep the payload as small as possible. XML is a great format but it's very verbose so given the option JSON may be better. Also, consider what transformations the RIA will have to do with the data. If the data can be used without any manipulation this can take a large load off of the client application.

Hopefully this has given you a taster of how a Silverlight RIA can consume real-time data and a few things to consider when doing so. I'm sure we are going to see more RTRIAs (Real-Time Rich Internet Applications) doing this as the real-time web continues to grow.

PHIL LEGGETTER
Blog: http://www.leggetter.co.uk
Twitter: http://twitter.com/leggetter

Podcast Available

MEET THE AUTHORS

ERIC NELSON

After many years of developing on UNIX/RDBMS (and being able to get mortgages) Eric joined Microsoft in 1996 as a Technical Evangelist (and stopped being able to get mortgages due to his new 'unusual job title' in the words of his bank manager). He has spent most of his time working with ISVs to help them architect solutions which make use of the latest Microsoft technologies - from the beta of ASP 1.0 through to ASP.NET, from MTS to WCF/WF and from the beta of SQL Server 6.5 through to SQL Server 2008. Along the way he has met lots of smart and fun developers - and been completely stumped by many of their questions! In July 2008 he switched role from an Application Architect to a Developer Evangelist in the Developer and Platform Group.

Developer Evangelist, Microsoft UK
Blog: http://geekswithblogs.net/iupdateable/
Twitter: http://twitter.com/ericnel
Email: eric.nelson@microsoft.com

MICHAEL FOORD

Michael Foord has been programming with Python since 2002 and is a Python core developer and Python Software Foundation member. Since 2006 he has been developing with IronPython, initially for Resolver Systems who have the largest IronPython codebase in the world in their Resolver One highly programmable spreadsheet application and since late 2009 as a freelance developer currently writing IronPython Silverlight applications for Comsulting. Michael is a Microsoft MVP and also the author of IronPython in Action for Manning Publications. Michael writes and blogs far more than is healthy for one individual and in the real world lives in Northampton with his wife Delia.

Book: http://www.ironpythoninaction.com/
Blog: http://www.voidspace.org.uk/blog

JOSH TWIST

Josh is an Application Development Consultant with Microsoft in the UK and works with a variety of customers, from small ISVs to large Enterprises. The Development Consultant role requires skills in all areas of the Microsoft Application Platform from architectural consulting and .NET development to debugging and performance tuning. Josh has recently been focussing on WPF and Silverlight and has acted as an advisor to Microsoft's Patterns and Practices group on both versions of their Composite Application Guidance.

Application Development Consultant, Microsoft UK
Blog: http://www.thejoyofcode.com/
Email: jtwist@microsoft.com

PAUL JACKSON

Paul is a Principal Technologist, where he creates developer-focused courseware and technical content. Prior to this, Paul has held various development roles over the last 16 years ranging from Software Engineer to Technical Architect. He has extensive development experience using many programming languages and methodologies now specializing in the .NET Framework and XAML based technologies. Paul's experience with technology over the years has been wide and varied, but more recently includes the .NET Framework 3.5 SP1 and 4, Windows Presentation Foundation (WPF), Windows Communication Foundation (WCF), Silverlight 1.0, Silverlight 2, Silverlight 3, ASP.NET MVC and Microsoft Surface development with WPF.

Principal Technologist, Swindon UK
Blog: http://compilewith.net

JOEL HAMMOND-TURNER

Joel works as a Technical Architect for Civica Ltd, and is currently bringing over 15 years experience of software development and a passion for both technology and elegance to designing a world-class Case Management system for the Health and Social Care markets using Silverlight, .Net and Oracle.

Outside of work he referees life with his wife and three young children in a home with more computers than is probably wise.

Joel can shoot video too, recently organising the filming of Scott Guthrie's recent Manchester Guathon.

Technical Architect, in4tek
Blog: http://bit.ly/goldennuggets
Twitter: http://twitter.com/Rammesses

ALEX MACKEY

I have just emigrated to Melbourne, Australia and am currently working as a Senior Developer for carsales.com.au. Prior to this move I worked as lead developer on a number of health care projects in the UK, Ireland and Middle east.

In the UK I founded and ran a .net user group (Devevening.co.uk) which combined beer, food and .net - this turned out to be a popular combination and DevEvening quickly grew. I have presented at a number of conferences and user groups such as DDD Belfast, Epicenter, Nxtgen, WebDD, Vbug London, Victoria .net user group and my own group to audiences of 5 to 120.

Senior Developer
Blog: http://www.simpleisbest.co.uk/

PETER O'HANLON

Pete started off his professional development career developing applications in C/UNIX, before moving to C++ and Windows back in the early 90s. After 10 years clearly stating that anybody who wanted to program in anything other than C++ was mad and that understanding ATL was necessary if you wanted to achieve coding Nirvana, Pete discovered C# when the .NET Beta for 1.0 first came out. More recently he discovered the joy of declarative code and WPF. At first, it seemed strange and alien to Peter, but he persisted, studying at the feet of the Maharishi Josh Smith and 10th Dan Adam Nathan. The veil was lifted, and he made the leap to writing properly data bound applications, transforming his development abilities so that he now cracks out applications like a leaping gazelle. Oh yes, Pete is a Code Project MVP and runs his own software business, where paperwork and meetings takes way too much time away from developing. Pete wonders why he spends so much time referring to himself in the third person.

Blog: http://peteohanlon.wordpress.com

CRAIG MURPHY

Craig Murphy is an author, developer, project manager, speaker, community evangelist and is a Microsoft MVP. He regularly writes articles product/book reviews: The Delphi Magazine, International Developer, ASPToday and the Developers Group magazine have published his work. He specialises in all things related to .NET, C#, Borland Delphi, XML/Web Services, XSLT, Test-Driven Development, Extreme Programming, agile methods and Scrum. His work with Scrum has been published in two academic text books. In his career to date, Craig has written cost estimating software for the oil and gas industry and asset valuation software for local councils and the Ministry of Defence. Craig is part of the team behind the DeveloperDeveloperDeveloper (DDD) events – he has been involved in locating speakers, preparing agendas, liaising with speakers, finding swag and being part of the DDD team that "makes things happen". He has a day-job, a wife and a son.

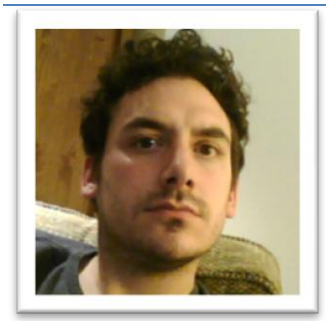
Systems Development Engineer at Sinclair Knight Merz, Edinburgh, UK
Blog: http://www.craigmurphy.com/blog
Email: web@craigmurphy.com

MIKE ORMOND

Mike joined Microsoft in 1997 after an unfortunate misunderstanding involving a sock puppet and some oven-ready chips. Prior to that he'd worked for sector-leading companies such as Mars Electronics and Mercury Interactive. He spends most of his time trying to keep abreast of web technologies and managing a nasty habit for classic car magazines. His dream job would be to web-enable a Porsche 914. Away from work Mike lives in the wilds of Wiltshire with his wife, baby daughter and an enormous tabby cat.

Developer Evangelist, Microsoft UK
Blog: http://blogs.msdn.com/mikeormond

PHIL LEGGETTER



Phil Leggetter is a Real-Time Web Software Consultant with over 9 years experience with real-time web technologies and methodologies (Yes, the real-time web has been around that long). In 2008 Phil formed his own company to provide real-time web, real-time data and social media software consultancy. His company specialises in the use of comet servers and Real-Time Rich Internet Application (RTRIA) development in Silverlight and Ajax and Flex/Flash.

Phil Leggetter Real-Time Web Software Consultant at Independent Software Solutions
Blog: http://www.leggetter.co.uk
Company: http://independentsoftwaresolutions.com
Email: phil@leggetter.co.uk

MIKE HADLOW

Mike Hadlow has been building business software since VB3. As a freelance .NET developer he likes to sneak into various public and private organisations, only to annoy them intensely with his outspoken opinions on how they really should be writing software. In his spare time he enjoys expressing those same opinions on his blog and in person at various developer events. He is also the author of a little known open source eCommerce application, Suteki Shop. His most important role, however, is human trampoline to his young children Leo and Yuna.

Director of Suteki Ltd, Brighton UK
Blog: http://mikehadlow.blogspot.com
Email: mikehadlow@yahoo.com

HADI HARIRI

Hadi Hariri is a software developer. His passions include software architecture and web development. Book author and frequent contributor to developer publications, Hadi has been speaking at industry conferences and user groups for over a decade. He is based in Spain where he lives with his wife and two sons and runs the Malaga .NET User Group.

Principal Consultant for iMeta Technologies
Blog: http://hadihariri.com/

GARY SHORT

Gary Short works for Developer Express as the Technical Evangelist on the frameworks team. He has a deep interest in technical architecture, especially in the areas of technical debt and refactoring. Gary is a C# MVP and gives presentations at user groups and conferences throughout the UK.
As well as C#, Gary also has an interest in dynamic languages such as Smalltalk, Ruby and Python as well as iPhone development using Objective-C

Evangelist, Developer Express
Blog: http://community.devexpress.com/blogs/garyshort/

TOM QUINN

Tom works as a Technical Architect for iMeta Technologies in Southampton. Tom started out as a C++ programmer, specialising in business to business e-Commerce applications. Tom later moved into the financial sector, using VB6 for his sins, building online applications targeted at IFAs. Tom moved into .Net development when version 1 was released and has continued to be focussed on the financial sector. Tom's main interests are TDD, Lean and pragmatic software delivery. Tom has also been known to scuba dive, but only in warmer climates.

Technical Architect for iMeta Technologies
Blog: http://blogs.imeta.co.uk/tquinn/Default.aspx