Issue 04 | JAN-FEB 2013

# DNC Magazine

www.dotnetcurry.com

Windows Azure Mobile Services and Web API

Real World Oauth using ASP.NET MVC

Microsoft Fakes framework in Visual Studio 2012

Web Essentials for Visual Studio 2012

Getting started with MVVM Light Toolkit for Windows 8

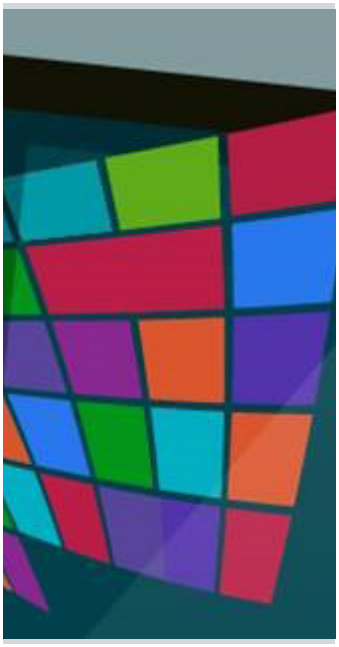
Storyboarding & Client Feedback using TFS 2012

Designing a Lossless Information System

Interview

PETE BROWN

## AZURE





## UNIT TESTING




## INTERVIEW



## MVVM




www.dotnetcurry.com
dnc mag

**Editor In Chief** Sumit Maitra
sumitmaitra@a2zknowledgevisuals.com

**Editorial Director** Suprotim Agarwal
suprotimagarwal@a2zknowledgevisuals.com

**Art & Creative Director** Minal Suprotim Agarwal
minalagarwal@a2zknowledgevisuals.com

**Advertising Director** Satish Kumar
business@dotnetcurry.com

**Writing Opportunities** Carol Nadarwalla
writeforus@dotnetcurry.com

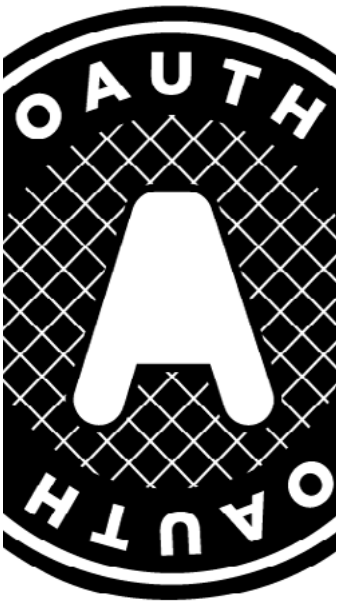
**Contributing Writers** Filip W, Piotr Walat, Gregor Suttie, Raj Aththanayake, Subodh Sohoni, Sumit Maitra

**Interview Feature** Pete Brown
Twitter @pete_brown

**Next Edition** 1st March, 2013
www.dncmagazine.com

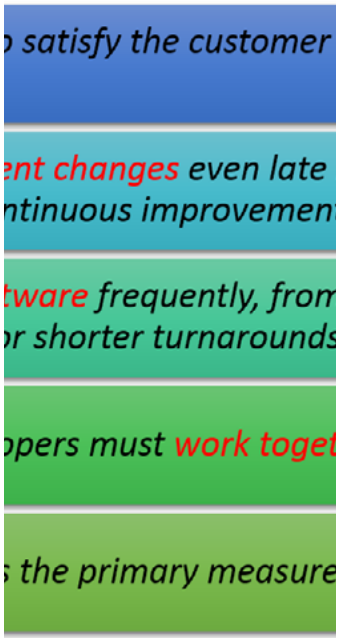
*Windows, Visual Studio, Azure, WinRT are trademarks of the Microsoft group of companies. 'DNC Magazine' is an independent publication and is not affiliated with, nor has it been authorized, sponsored, or otherwise approved by Microsoft Corporation*

## OAUTH






## TFS 2012






## VISUAL STUDIO




## DB DESIGN




Dear readers, wish you all a Very Happy and Prosperous New Year. 2012 was a fast paced year for both Web and Desktop Client technology on the Microsoft stack. Lots of firsts happened like the Azure SDK development going open source on Github, Microsoft releasing major chunks of ASP.NET as OSS, launch of Visual Studio 2012 and then of course we had the Windows 8 launch and Microsoft's first PC hardware - the SurfaceRT.

In the New Year, our resolution is that we'll continue to bring fresh and cutting edge development technology content for you. This issue has a platter of new topics, starting with Azure Mobile services by our first time contributor and Web API guru Filip W. We also have a nice introduction to MVVM Light by new contributor Piotr Walat.

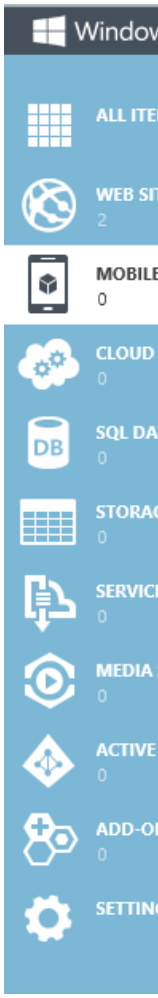
For our interview special, we bring you 'Pete Brown Unplugged'! We all know Pete as the Author of several Silverlight and XAML books and as Developer Evangelist for client side technologies at Microsoft. However in the interview, we'll discover a lot about his passions, hobbies, Microsoft and OSS.

To round off, we have some awesome contributions from Subodh Sohoni, Raj Aththanayake and Gregor Suttie among others.

Sumit K. Maitra
Editor in Chief

# WINDOWS AZURE MOBILE SERVICES AND WEB API

## A MATCH MADE IN HEAVEN




**FILIP W. DEMONSTRATES AN EXCITING NEW CLOUD SERVICE THAT GETS YOUR APP UP AND RUNNING WITHOUT HAVING TO WORRY ABOUT SCALING YOUR DATABASE!**

What if someone told you, in your next application you don't have to think about the database at all? What if they told you, that days of worrying about fast, scalable data access layer are gone and that in a few simple clicks, you can have a schema-free cloud storage which you can share between your applications?

Would you believe them or would you call them crazy?

Because you can have it all here, now, with Azure Mobile Services; and by combining its REST API flexibility with a custom Web API wrapper, you can build literally any application you want, with less effort than you could have dreamed of.

### SO WHAT IS THIS AZURE MOBILE SERVICES?

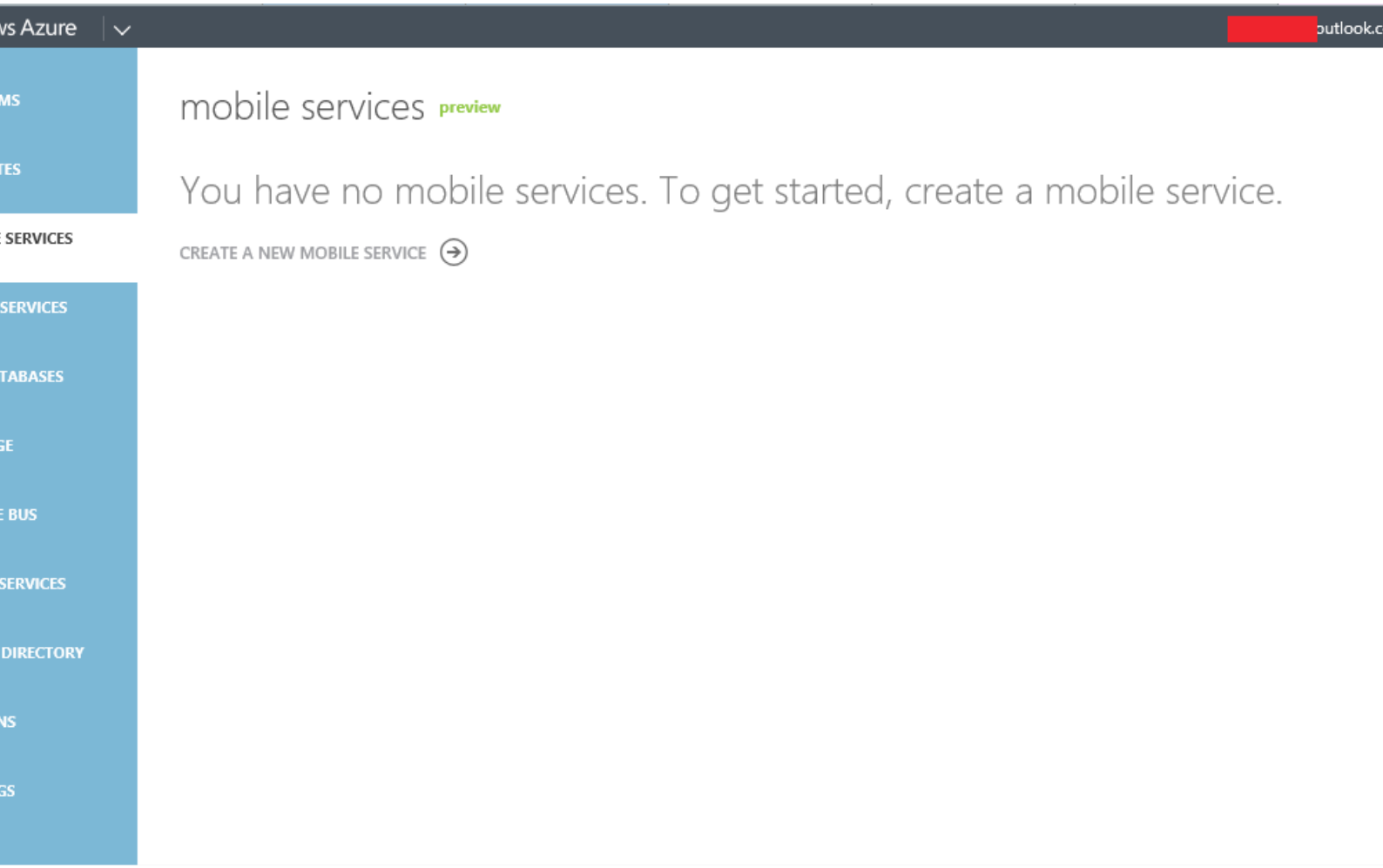
Azure Mobile Services, or ZUMO, one of the latest additions to the Windows Azure IaaS family (launched in August 2012), is a service aimed at providing developers with scalable, hassle-free backend and storage functionalities.

While its name suggests that it is a mobile service, that's actually not entirely accurate. Alongside the ZUMO SDKs for Windows 8 and iOS, it also exposes a very slick REST API which can be consumed by any device. And therein lies the true power of ZUMO - as its fantastic capabilities can be used to fuel any application.

And that's precisely what we will be focusing on in this article. ZUMO offers many rich functionalities through its SDKs, we will explore the REST API and how we can build a repository for our apps around it.

### HOW DOES WEB API FIT INTO THIS?

By now, you are probably wondering, what does Web API have to do with all this? And rightfully so - it doesn't have anything to do with ZUMO. In fact, since ZUMO has a REST API, you do not need Web API for anything. You can just query ZUMO directly using your favorite web request technique, from any platform you wish.




This is all true.

However, layering a middleman in between, in the form of a Web API service opens tremendous possibilities. Think of it as a typical n-tier (3-tier in this case) architecture:

- ZUMO providing data layer
- Web API providing the service layer
- Whatever client you have, providing the presentation layer (HTML5, XAML, Cocoa, etc.)

You can now support multiple platforms and multiple types of applications with the same common logic stacked in your Web API service layer. Since everything is based on HTTP you can scale extremely easily. Moreover, through Web API, you can fully embrace HTTP and benefit from all its extra freebies such as client caching.

## GETTING STARTED WITH AZURE MOBILE SERVICES

In order to be able to proceed with the examples shown in this article, you need to quickly head over to www.windowsazure.com and sign up for a 3-month free trial.

The whole thing shouldn't take more than 5 minutes. Azure Trial needs a USD Credit Card transaction capability, however there won't be a charge after your trial expires; rather the services that do not fall under the 'free' category will stop functioning.

Then, once everything is set up and you have your free account, head over to the Azure Management Portal and set up Azure Mobile Services.

You should be ready to go.

> *"... You can just query ZUMO directly using your favorite web request technique, from any platform you wish..."*

## SETTING UP A ZUMO SERVICE

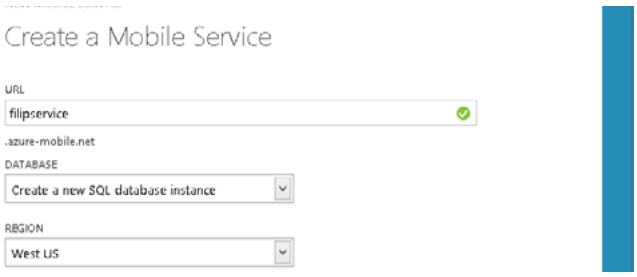
Remember, ZUMO storage is schema-free - which in other words, means "take whatever you have and just dump into the cloud." It has the concept of tables, and you can think of them as API endpoints - each table represents some resource - i.e. Cars, Products, Teams or what have you.

Now, since it's schema-free, it makes it really easy to work with; all you have to do when defining your application's backend data storage is to identify those main resources which need to be represented as ZUMO's tables. If you are used to working with NOSQL types of databases, you will definitely be familiar with the concept. The difference in ZUMO is that, under the hood, there still is a SQL Server that hosts your data - it's just that ZUMO abstracts it completely away and dynamically modifies the table's schema to facilitate the type of data that's getting pushed into a given table

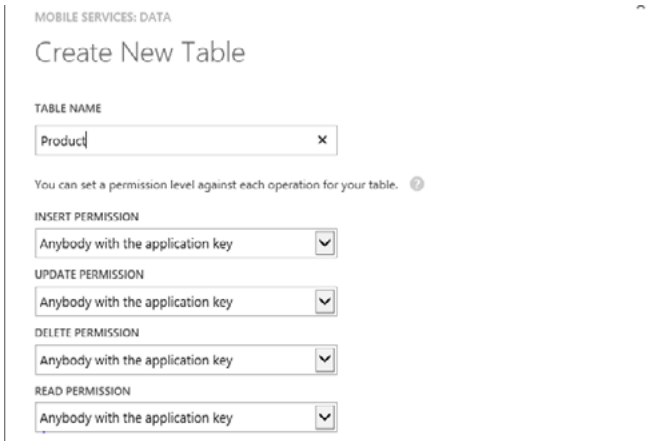
*Note: once your development is over, and you push your app to production, you really should disable this dynamic schema for performance considerations).*

Let's set up a sample service:

For demo purposes, we will have a simple Product manager. We will build a Web API service, wrapped around a ZUMO-powered repository implementation.




Once the service is created, we create the single table that we are using for the Demo. Note we define the Table only. The Schema is managed automatically.




## CREATING THE WEB API SERVICE

Let's create a new MVC4, Web API project. In here, we will implement a generic repository against our newly created ZUMO service. You see where we are going with that?

Exactly!

I told you before we could forget 'against the DB', and that's what we will do. Our repository, although from the client code perspective indistinguishable from any other repository interface implementation (making the client code completely unaware of what's going on behind the scenes), will actually be sitting in the Azure cloud.

But first things first. The model and a base entity:

```
public abstract class Entity
{
 public int? id { get; set; }
}

public class Product : Entity
{
 public string Name { get; set; }
 public decimal Price { get; set; }
 public int StockCount { get; set; }
}
```

The fact that we introduced an abstract base type will simplify querying by ID (get, delete) against a generic repository implementation.

A typical repository interface would look like this:

```
public interface IRepository<T> where T : Entity
{
 void Add(T entity);
 void Delete(int id);
 T Get(int id);
 IEnumerable<T> GetAll();
 void Update(T entity);
}
```

And you can imagine a typical controller built around this repository:

```
public class ProductController : ApiController
{
 private IRepository<Product> _repo;
 public ProductController(IRepository<Product> repo)
 {
  _repo = repo;
 }
 public IEnumerable<Product> Get()
```

```
 {
  return _repo.GetAll();
 }
 public Product Get(int id)
 {
  return _repo.Get(id);
 }
 public void Post(Product product)
 {
  if (ModelState.IsValid)
   _repo.Add(product);
  else
   throw new HttpResponseException(HttpStatusCode.
BadRequest);
 }
 public void Put(Product product)
 {
  if (ModelState.IsValid)
   _repo.Update(product);
  else
   throw new HttpResponseException(HttpStatusCode.
BadRequest);
 }
 public void Delete(int id)
 {
  _repo.Delete(id);
 }
}
```

## IMPLEMENTING REPOSITORY AROUND AZURE MOBILE SERVICES

In order to integrate ZUMO into our app, we will implement the repository interface with the use of HttpClient and ZUMO's REST API.

The API is very well documented at http://msdn.microsoft.com/en-us/library/windowsazure/jj710108.aspx but it's very intuitive even if you are too lazy to read any API reference. As expected from a REST API, you get to deal with the usual HTTP verbs only:

- GET https://[your_service].azure-mobile.net/tables/[table_name]/ - to get entries, supports OData queries
- POST https://[your_service].azure-mobile.net/tables/[table_name]/ – to add a new entry
- PATCH https://[your_service].azure-mobile.net/tables/[table_name]/[id] – to update an entry
- DELETE https://[your_service].azure-mobile.net/tables/[table_name]/[id] – to delete an entry

With that said, here is our repository (or rather, it's first part):

```
public class ZumoRepository<T> : IRepository<T> where T
: Entity
{
 private readonly HttpClient _client;
 private readonly string _key = [YOUR-API-KEY];
 private readonly string _address = "https://[YOUR-
SERVICE].azure-mobile.net/tables/" + typeof(T).Name.
ToString();

 public ZumoRepository()
 {
  _client = new HttpClient();
  _client.BaseAddress = new Uri(_address);
  _client.DefaultRequestHeaders.Add("X-ZUMO-
APPLICATION", _key);
  _client.DefaultRequestHeaders.Accept.Add(new
    MediaTypeWithQualityHeaderValue("application/
json"));
 }
 //omitted for brevity
}
```

Upon creation of the repository, we will create a new HttpClient, and point it to our ZUMO API. We will use this common setup across different repository methods so it only makes sense to have it all set up from the get go. Notice that we will be sending our API key alongside each request, in the headers.

The individual repository methods are implemented below:

```
public IEnumerable<T> GetAll()
{
 var resultTask = _client.GetStringAsync(string.Empty);
 var obj = JsonConvert.DeserializeObject<IEnumerable<T>>
(resultTask.Result);
 return obj;
}

public T Get(int id)
{
 var resultTask = _client.GetStringAsync("?$filter=Id eq
" + id);
 var obj = JsonConvert.DeserializeObject<IEnumerable<T>>
(resultTask.Result);
 return obj != null ? obj.FirstOrDefault() : null;
}

public void Add(T obj)
{
 var responseTask = _client
  .SendAsJsonAsync<T>(HttpMethod.Post,string.Empty,obj);
 if (!responseTask.Result.IsSuccessStatusCode)
 {
  throw new HttpResponseException(responseTask.Result.
StatusCode);
 }
}

public void Update(T obj)
{
 var responseTask = _client.SendAsJsonAsync<T>(new
HttpMethod("PATCH"), obj.id.ToString(), obj);
 if (!responseTask.Result.IsSuccessStatusCode)
```

```
 if (!responseTask.Result.IsSuccessStatusCode)
  throw new HttpResponseException(responseTask.Result.
StatusCode);
}

public void Delete(int id)
{
 var responseTask = _client.DeleteAsync(_client.
BaseAddress+"/"+id.ToString());
 if (!responseTask.Result.IsSuccessStatusCode)
  throw new HttpResponseException(responseTask.Result.
StatusCode);
}
```

So, in a nutshell, we just query the remote ZUMO API, and deserialize the returned JSON objects into our Product type or serialize the outgoing input objects into JSON for ZUMO to consume. As mentioned below, Azure Mobile Service's GET method supports OData querying so you can get as creative as you want with your specific Get implementations.

Notice that we run everything synchronously and block on the execution of the tasks - while we don't have to do that, and could have async methods with Task return types, we want to keep the flexibility of adhering to the original repository interface, and that was synchronous to begin with.

One final thing is that our repository uses a custom extension method SendAsJsonAsync<T>. It's just a helper to provide cleaner code:

```
public static class HttpClientExtensions
{
 public static Task<HttpResponseMessage>
SendAsJsonAsync<T>(this HttpClient client, HttpMethod
method, string requestUri, T objectValue)
 {
  if (client == null) throw new ArgumentNullException(
"client");
  if (requestUri == null) throw new ArgumentNullExcept
ion("requestUri");
  if (objectValue == null) throw new ArgumentNullExcep
tion("objectValue");

  var serializedObject = JsonConvert.
SerializeObject(objectValue, new
JsonSerializerSettings()
  {
   NullValueHandling = NullValueHandling.Ignore
  });
  var request = new HttpRequestMessage
  {
   Method = method, RequestUri = new Uri(client.
BaseAddress+"/"+requestUri)
  };
request.Content = new StringContent(serializedObject,
Encoding.UTF8, "application/json");
  return client.SendAsync(request);
 }
}
```

## RUNNING OUR CODE

Of course you need to inject the repository implementation into the controller somehow. You can use your favorite IoC framework for that, in this example, for simplicity reasons, I'll manually new it up in the constructor.

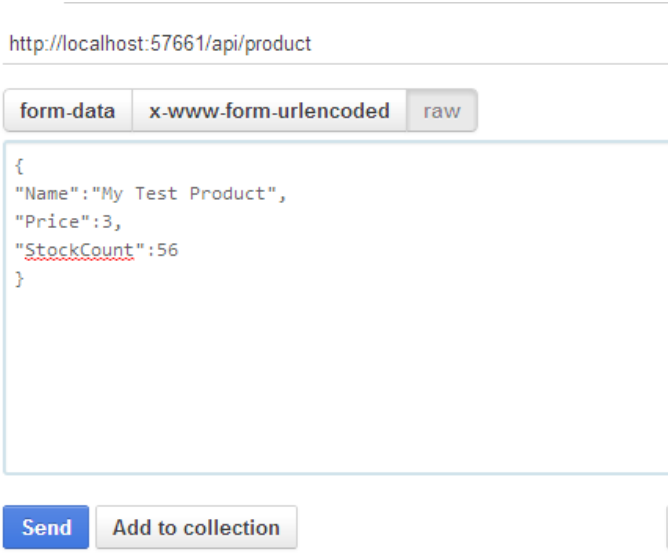
```
public ProductController() : this(new
ZumoRepository<Product>())
{}
```

You can now walk up to our Web API and do any of the typical API operations:

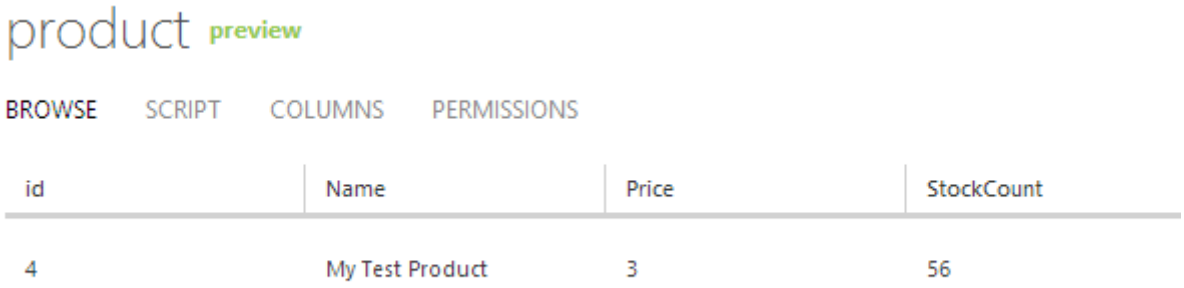
- GET /api/product
- GET /api/product/[id]
- POST /api/product (to add item)
- PUT /api/product (to update item)
- DELETE /api/product/[id] (to remove item)

In the background, our ZUMO repository will hop over to the cloud and perform the data operations there. For example, adding new products via our Web API, and see them appear in the ZUMO management portal.

We can use tools like Fiddler or the Postman plugin for Chrome/ Firefox to POST some sample data.




If your formatter is set to return XML, it will return the following:


In a similar way, you can perform an HTTP PUT to update the resource (in the background it gets translated it into an HTTP PATCH for ZUMO) or HTTP DELETE to remove it.

There are many ways to go from here.

You could use our ZUMO repository to simply build a SPA type / modern web app using Web API only. Alternatively, you could build ANY client app, and use the Web API as your service provider; use it to provide any business logic, handle authentication and then let it jump to ZUMO whenever it performs data operations.

Either way, let's reiterate what we said in the very beginning - a typical n-tier application would have three layers:

1. Presentation (could be any type of app)
2. Service layer (handles everything related to business logic)
3. Data layer

With ZUMO and Web API working side by side, you can easily cover 2 and 3, and worry only about how your app should be presented!

## SUMMARY

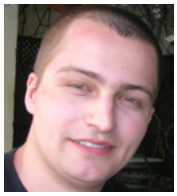
I really recommend you take a deeper look at Azure Mobile Services. Contrary to popular belief, it's not only a service aimed at mobile applications. You can really do a lot with it, and in many cases it can be a very viable, scalable, replacement for your database.

And with Web API acting as an intermediary, you can build distributed applications with extreme ease.

One thing can be said for certain - it really is a great time to be a .NET developer.

P.S. If you wish to learn more about Azure Mobile Services, make sure to follow @joshtwist on Twitter. ■

Source for the Sample application is available on Github at http://bit.ly/dncmag-zwa



Filip works as a Senior .NET Developer at Climax Media (http://www.climaxmedia.com/) in Toronto, Canada. He specializes in developing large, multilingual corporate websites and have worked on a number of projects in cooperation with large partners (Microsoft, Accenture/Avanade, IBM), in many corners of the world (Canada, Switzerland, Finland, Poland, Scotland). He is a member of the ASP.NET Web API advisory board and blogs about Web API at http://www.strathweb.com. You can find him on Twitter at @fillip_woj

# MICROSOFT FAKES FRAMEWORK IN VISUAL STUDIO 2012

*Raj Aththanayake introduces the Microsoft Fakes framework in Visual Studio 2012 Ultimate and gives a quick run-down on how to use Fakes, Shims, Stubs and Observers to build a robust test harness for your .NET applications*

In Unit Testing/Test Driven Development (TDD) Isolating dependencies, sometimes referred to as "mocking objects" or "faking objects", is a norm. There are many books/articles that have been written on this topic. There are also great frameworks out there, which allow us to isolate dependencies without having to use any hand written mocks/fakes. In VS 2012 Ultimate, the new Microsoft Fakes framework takes mocking to a new level. In this article, I will explain the features of the framework. Also please remember that currently the Fakes framework is only available in Visual Studio Ultimate edition.

## SOME HISTORY BEHIND THE FAKES FRAMEWORK

If you have used Microsoft Moles before, you will find the Fakes framework has a very similar usage workflow. Moles were originally designed to support the development of Microsoft Pex , which is another great Unit Testing tool by Microsoft Research. Microsoft Research did not use other isolation frameworks because, they wanted something powerful that could mock pretty much anything, was simple and had absolutely no overhead of using it. As you would imagine, most isolation frameworks use dynamic proxies to generated mock objects, which have some overhead of using it. Moles provided a simple type safe detour framework with very clear semantics. In a nutshell, Moles replaces any .NET method with a delegate. In VS2012, with some improvements, Moles framework became the Microsoft Fakes framework.

## HARD TO TEST CODE

As you probably know, static methods by nature, are harder to test. A static method itself is easy to test, but the code that is calling the static methods become harder to test because it is tied into the type of the class and thus cannot be easily replaced by something else.

For example, let's say that we want to test a piece of code that throws an exception, when the current date reaches 21st of December 2012. (Reminds me of the movie "2012" but it is not doomsday. If you are reading this article, this date has passed already and we are all good :) ).

```
public static void DoomsDay()
{
 if (DateTime.Now == new
DateTime(2012, 12, 21))
  throw new Exception("Boom!");
}
```

Let's say I want to test this method. I want the exception to be thrown when the current date equals to 21st of December 2012. Assuming today's date is not 21st of December, the test would not throw an exception. So how can we make the test to fail with the "Boom!" exception? We need to get hold of DateTime.Now property. Previously I mentioned that "replacing any .NET method with a delegate". For example, I should be able to easily replace calls to the DateTime.Now with a delegate as below.

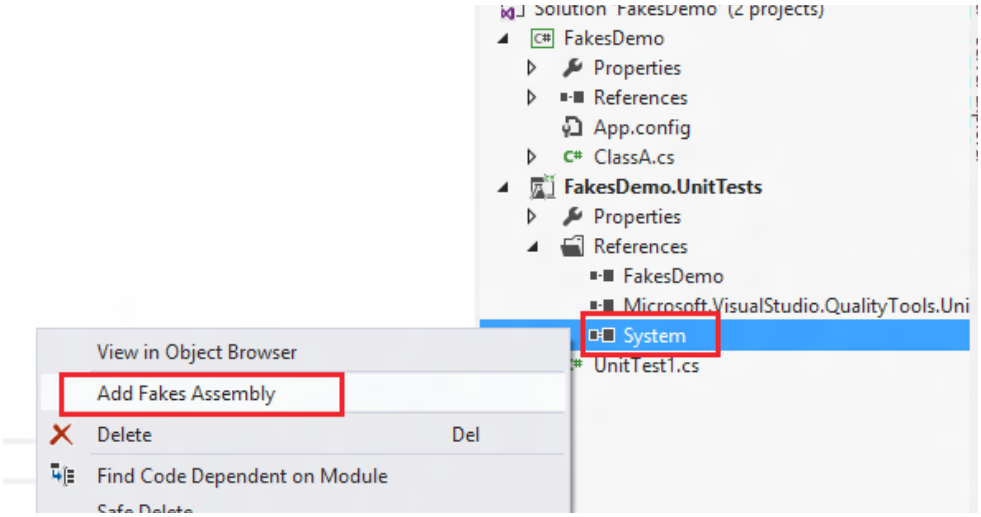
```
DateTime.Now = ()=> new
DateTime(2012, 12, 21);
```

This code cannot be compiled because in .NET DateTime.Now property cannot be assigned to a delegate. However using the new Fakes framework, you can replace this property with a delegate.

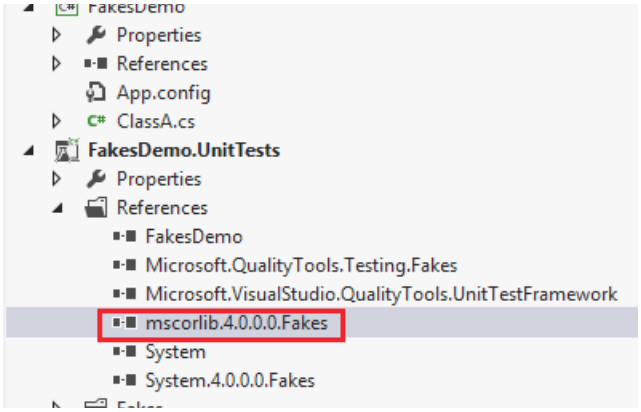
I will revisit this example a bit later, but now let's look at how to enable Fakes within your project.

## ENABLING FAKES

Enabling Fakes is about creating a fake assembly that allows you to use (fake) types that replace .NET types with delegates. In this case, it is the System.dll, which wraps the mscorlib.dll that contains the DateTime object.




Once you "Add Fakes Assembly", there are few new files/assemblies that get added to the test project as seen below.




Note that mscorlib.4.0.0.0.Fakes assembly contains a special type called Shims. Shims are strongly typed wrapper that allows you to replace .NET types with delegates.

## USING THE SHIM FOR DATETIME CLASS

We just generated the Fakes library that contain Shims and now we can look at replacing the DateTime.Now with a delegate. In here, instead of using the real DateTime.Now property, you can use the Shim DateTime as below.

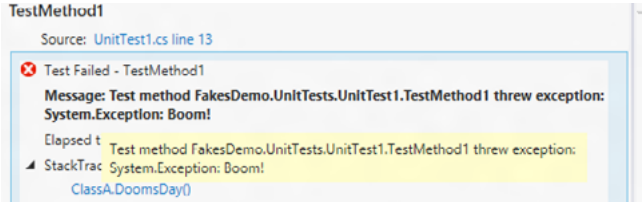
```
[TestMethod]
 public void TestMethod1()
 {
  ShimDateTime.NowGet = ()=> new DateTime(2012, 12,
21);
  ClassA.DoomsDay();
 }
```

Notice that there is a naming convention. You need to prepend the word **"Shim"** to the real .NET type and call the appropriate property that uses the delegate. Now if I run the test, my test fails! But if you look at the exception, it failed for a different reason. The complete exception is below.

Test method FakesDemo.UnitTests.UnitTest1.TestMethod1 threw exception: Microsoft.QualityTools.Testing.Fakes.Shims.ShimInvalidOperationException: A ShimsContext must be in scope in order to register shims. Use the snippet below to fix your code.-- C#:using Microsoft.QualityTools.Testing.Fakes;using(ShimsContext.Create()){ // your test code using Shims here}--

The Shims require running their own context. This guarantees the level of isolation that is required by Shims, have no impact on other domains whatsoever. This means that whenever you use a Shim, you need to create a context that Shim persists and destroy the Shim when the context finishes execution. You can create the ShimContext as below.

```
[TestMethod]
public void TestMethod1(){
 using(ShimsContext.Create()) {
  ShimDateTime.NowGet = () => new DateTime(2012, 12,
21);
   ClassA.DoomsDay();
 }
}
```



As you see in the screenshot, we hit the "Boom" exception. The *ShimDateTime.NowGet* replaces the real DateTime.Now value with our own DateTime value, which we have provided via the delegate.

Before we see more examples, let's see what's really happening under the hood. We set a breakpoint on the delegate *new DateTime(2012, 12, 21);* and re-run the test. Once the breakpoint hits, you should see the stack trace as below.

```
FakesDemo.UnitTests.dll!FakesDemo.UnitTests.UnitTest1.
TestMethod1.AnonymousMethod__0() Line 16
mscorlib.dll!System.DateTime.Now.get() + 0xa9 bytes
FakesDemo.dll!FakesDemo.ClassA.DoomsDay() Line 13 + 0x8
bytes
FakesDemo.UnitTests.dll!FakesDemo.UnitTests.UnitTest1.
TestMethod1() Line 17 + 0x6 bytes
[Native to Managed Transition]
....
```

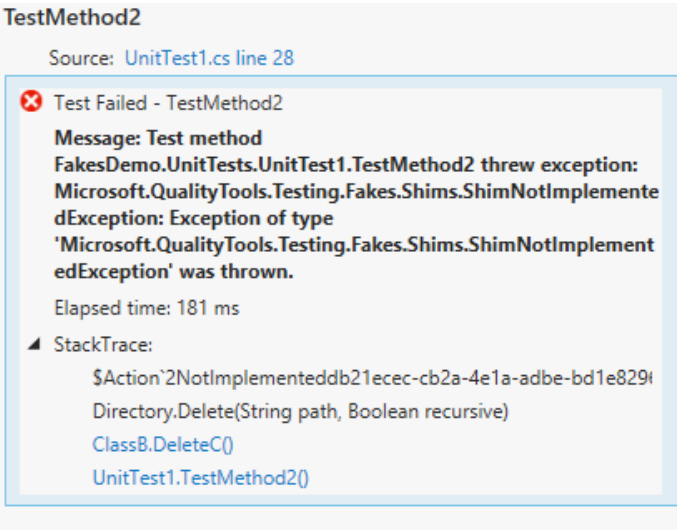
Once the code executes the DateTime.Now property in mscorlib.dll, it has taken detour to an anonymous method specified by the delegate. Using .NET reflector, if you take a closer look at the implementation of **ShimDateTime.NowGet,** you can see the **ShimRunTime** replaces the DateTime.Now -> Get property with the delegate, which we have set in the test method.

```
public static FakesDelegates.Func<DateTime> NowGet
{
  [ShimMethod("get_Now", 24)] set
  {
    ShimRuntime.SetShimPublicStatic((Delegate) value,
     typeof (DateTime), get_Now",
     typeof (DateTime), new Type[0]);
  }
}
```

So far we have discussed an example of a Shim. Shims allow you to replace methods that are Static, Non-override-able, or even private methods (I called them all unfriendly) with delegates. It re-writes the code at runtime. With Shims, you can even fake the deletion of your C:\ drive without actually deleting it. You can also generate code and inject them into the system. For example, using ShimDirectory. BehaveAsNotImplemented() method you can inject the code to throw NotImplementedException.

```
//Warning - Do not attempt this outside the test con-
text
public static void DeleteC()
{
 Directory.Delete(@"C:\TEMP", true);
}
```

```
[TestMethod]
public void TestMethod2()
{
 using (ShimsContext.Create())
 {
  ShimDirectory.BehaveAsNotImplemented();
   ClassB.DeleteC();
 }
}
```




## STUBS

In Unit Testing, stubs provide canned answers to the System Under Test (SUT). As Martin Fowler puts it:

*"**Stubs** provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'."*

The Fakes framework also has the ability generate stubs; these have more to do with virtual types. For example, Stubs can be used to fake virtual types such as interfaces, abstract classes, virtual methods etc. Stubs cannot be used in non-virtual context such as private, static or sealed types.

Let's look at an example of a Stub.

*System Under Test (SUT):*

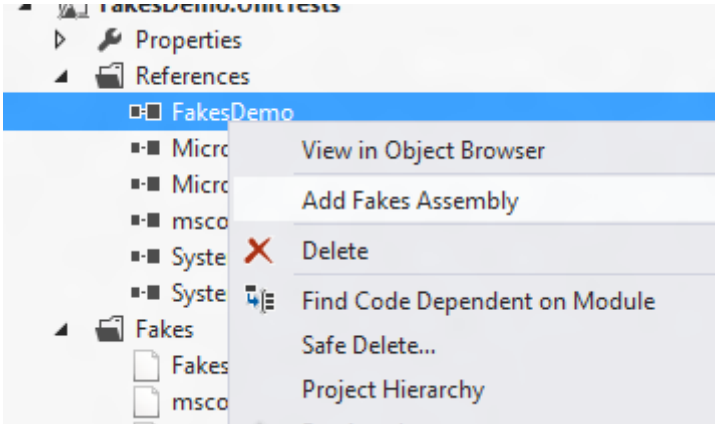
```
public interface ILogger{
 bool ShouldLog();
 void Log(string message);
}

public class ClassA
{
  public string GetName(ILogger logger)
```

```
 {
  if (logger.ShouldLog())
  {
   logger.Log("log something");
   return "some name";
  }
  return string.Empty;
 }
}
```

*Test method:*

In order to use stubs in our test, we need to add a Fakes assembly for the library that contains the virtual type/interface. In our previous Shims example, we add the Fakes assembly for the System.dll. In this case, it is the FakesDemo.dll. You only have to do this once, and then every time when you build the application, the Fakes assembly gets auto generated.




Once we add a Fakes assembly, **FakesDemo.Fakes.dll** gets generated. This fakes assembly contains the **Stub types** that we can use to replace the ILogger implementation. Please see below.

```
[TestMethod]
public void TestMethod1()
{
 //Arrange
 var stubLogger = new StubILogger
 {
  ShouldLog = () => true
 };
 var sut = new ClassA();
 //Act
 var result = sut.GetName(stubLogger);
 //Assert
 Assert.IsTrue(!string.IsNullOrEmpty(result));
}
```

*StubILogger* is a type that was generated for you by the Fakes Framework. *ShouldLog* is a property delegate (Func<T>) within the *StubILogger*.

If you are interested in how the FakesDemo.Fakes.dll and the stub types get generated in the first place, it is done by the Microsoft.QualityTools.Testing.Fakes assembly. This assembly generates a Fakes project (i.e f.csproj) and the associated C# classes (i.e f.cs) and compiles them into a dll (i.e FakesDemo.Fakes.dll). The source code for the auto generated class and the other artefact can be found in the test project's obj/Debug/Fakes/...

Below is a trimmed down version of the auto generated class StubILogger

```
namespace FakesDemo.Fakes
{
 public partial class StubILogger
  {
  /// <summary>Sets the stub of ILogger.ShouldLog()</
summary>
  public mqttf::Microsoft.QualityTools.Testing.Fakes.
FakesDelegates.Func<bool> ShouldLog;
  /// <summary>Sets the stub of ILogger.ShouldLog()</
summary>
  bool fd::FakesDemo.ILogger.ShouldLog()
  {
   mqttf::Microsoft.QualityTools.Testing.Fakes.Stubs.
IStubObserver ___observer
    = ((mqttf::Microsoft.QualityTools.Testing.Fakes.
Stubs.IStubObservable)this).InstanceObserver;
   if ((object)___observer != (object)null)
   {
    mqttf::Microsoft.QualityTools.Testing.Fakes.
FakesDelegates.Func<bool> ___currentMethod =
((fd::FakesDemo.ILogger)this).ShouldLog;
    ___observer.Enter(typeof(fd::FakesDemo.ILogger),
(global::System.Delegate)___currentMethod);
   }
   mqttf::Microsoft.QualityTools.Testing.Fakes.
FakesDelegates.Func<bool> ___sh = this.ShouldLog;
   if ((object)___sh != (object)null)
  return ___sh.Invoke();
   else
   {
    mqttf::Microsoft.QualityTools.Testing.Fakes.Stubs.
IStubBehavior ___behavior
                  = ((mqttf::Microsoft.QualityTools.
Testing.Fakes.Stubs.IStub)this).InstanceBehavior;
    return ___behavior.Result<global::FakesDemo.Fakes.
StubILogger, bool>(this, "FakesDemo.ILogger.ShouldLog");
   }
}
```

The highlighted text shows 2 important parts

• We have a public delegate/Func field ShouldLog, which can be used to replace with our own delegate.

• private FakesDemo.ILogger.ShouldLog() method, which has the implementation of the invocation of the ShouldLog delegate.

```
mqttf::Microsoft.QualityTools.Testing.Fakes.FakesDelegates.
Func<bool> ___sh = this.ShouldLog;
if ((object)___sh != (object)null)
 return ___sh.Invoke();
```

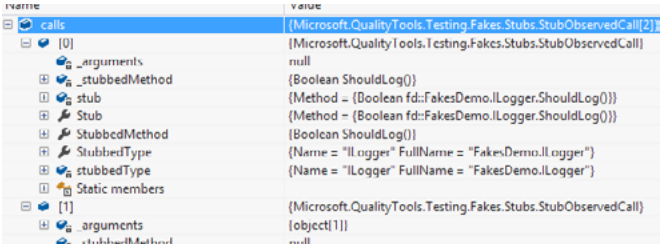
As you see above, once the ShouldLog delegate is not null, it invokes the delegate. Previously explained Shims does the same thing, but it all happens dynamically at run time.

## OBSERVERS

Observers allow you to record the calls on stubs. Fakes framework ships its own observer, which allows you to record every call, every argument made in the stub. This is similar to what you have seen in standard isolation frameworks, where you would verify (i.e mock.Verify(x => x.SomeMethod()) the interaction of a method call.

```
[TestMethod]
public void TestMethod1()
{
 //Arrange
 var stubLogger = new StubILogger {ShouldLog = () =>
true};
 var sut = new ClassA();
 var observer = new StubObserver();
 stubLogger.InstanceObserver = observer;
 //Act
 sut.GetName(stubLogger);
 var calls = observer.GetCalls();
}
```

If you see the debug information of the "calls" variable, you can see the following.




There are two calls made – *ShouldLog()* and *Log(message)*

Each call has arguments (if any), stub method, stub MethodInfo, and stub method type etc.

These are valuable information if you want to verify a certain method has been called with certain parameters. Some mock frameworks allow us to verify the number of occurrence of each method call, but I could not see a similar feature in the built-in StubObserver. Also important to note that you can write your own Observer that allows you to extend additional features.

## USING SHIMS VS. STUBS AND RECOMMENDATIONS

As I mentioned before, Shims are used mainly to test the untestable code. You find more of these code third party components that contains lots of statics, privates etc. Often this implies there is some code smell. It is recommended to use Stubs, because the virtual types are extendable, replaceable and easily testable. The other aspect is that if you see the code that is hard to test, and you have no confidence of changing code, you can use Shims to write integration type tests and refactor the code incrementally to make the code testable.

## SUMMARY

We looked at some of the features of the new Microsoft Fakes framework.

The key take away is that you can replace any .NET method with a delegate using the Fakes framework. The statically typed detour framework provides some powerful features, which allow you to stub hard-to-test methods such as statics and privates.

With Shims, the detour/delegation occurs dynamically at run time. ShimContext ensures that there is no side effect by creating a new app domain for the entire test context which detour occurs.

Stubs are lightweight wrappers which allow you to stub virtual types. Fakes Framework tools auto generate necessary code and compiles them into fakes dll, which you can use to stub types without having to hand craft them.

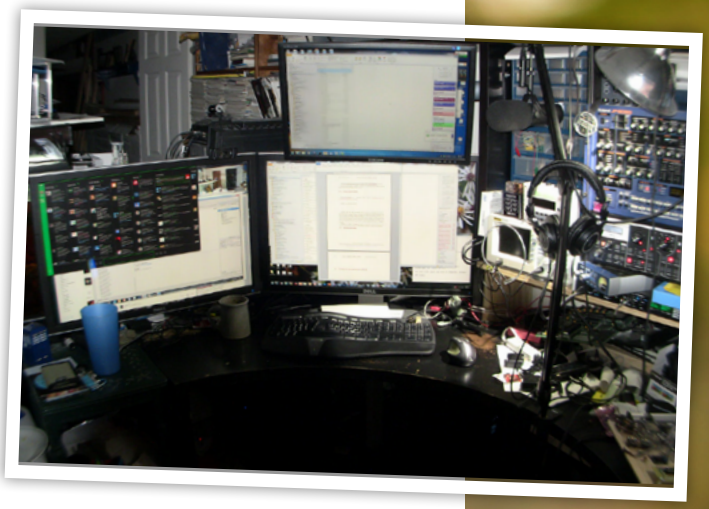
Observers are a great way to record stub interactions in the SUT (System Under Tests) and verify them within the test. ■

Code for the Sample Application can be downloaded from http://bit.ly/dncm-04-vsfgh

Raj Aththanayake is a Microsoft ASP.NET Web Developer specialized in Agile Development practices such Test Driven Development (TDD) and Unit Testing. He is also passionate about technologies such as ASP.NET MVC. He regularly presents at community user groups and conferences. Raj also writes articles in his blog http://blog.rajsoftware.com. You can follow Raj on twitter @raj_kba

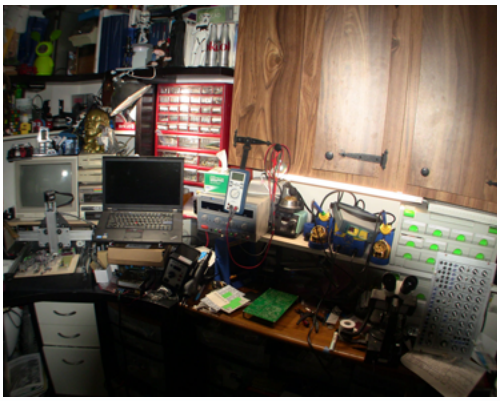

Follow us on
Twitter
@dotnetcurry

# Interview With Pete Brown



Dear readers, we continue our (three issues old!) tradition of featuring a prominent Community Member in the DNC Magazine. In our fourth issue, we are very happy to host the multi-talented Pete Brown from Microsoft! Pete with his wide range of interests is not just a .NET community member but a member of various engineering communities. Pete is well known in the .NET community as an Author of multiple XAML/Silverlight books (including the new Windows 8 XAML in Action) and a long time Developer Evangelist and community guy at Microsoft. However, he is active in robotics circles with his various experiments, a contributor to the Coding4Fun community as well as a Gadgeteer, Netduino, ARM native, and a MIDI (Musical Instrument Digital Interface) and synthesizer enthusiast. Well if you think that's a lot, we are just getting started, but instead of us spoiling the fun, let's hear from Pete himself.

## Take a look around my room..



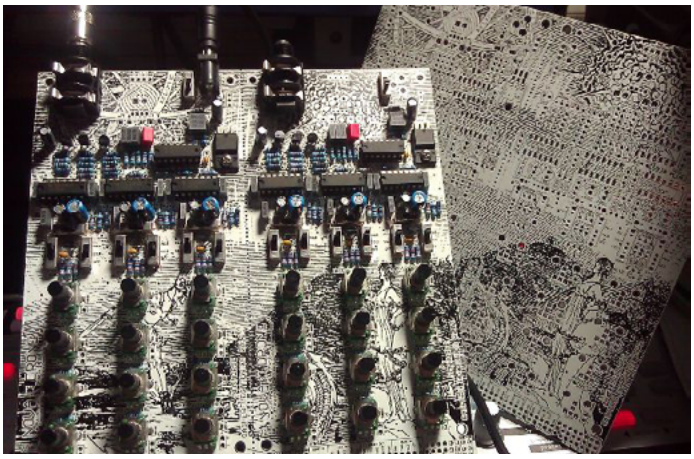
**DNC: Hello Pete, welcome to the DNC Magazine and thanks for taking time out for this interview. To start off with, what are 'blinky light' projects that you are working on currently or have just completed?**
**PB:** My favorite projects are those that combine modern technology with retro technology, or modern technology used for sound synthesis and control. My most recently completed project is a Nintendo gamepad interface for the .NET Gadgeteer. I was able to find the connectors (and even new controllers) online and then design a PCB around them. I posted some information on it to the

# Pete Brown

## Up, Close & Personal

GHI Electronics Forums.

I recently completed a Nova Drone kit, and posted a recording of its noise to my SoundCloud account. I have a lot of other projects in-progress, most of which include blinky lights. I have an x0xb0x kit that I have almost completed building. My cabinet has about a million other projects in it.



Finally, I'm working on some interesting ways to control MIDI synthesizers from Windows 8 (and Windows RT) using WiFi. I've been writing C code on ARM microcontrollers to be the synth interface on the other end of WiFi. There are some real possibilities here, as no one really wants to hang a long cable off of their tablet if they don't have to.

**DNC: That's an amazing breadth of 'side-projects' and it brings us to our next question - What got you interested in Electronics and Robotics? Was it a single incident or was it a sequence of events?**
**PB:** I blame Netduino! One time on a visit to the Microsoft campus in Redmond, I visited Clint Rutkas (Coding4Fun) and he gave me an original Netduino to play with. I went back to the hotel there in Bellevue, made the LED blink, and then realized I wanted to do something more. I threw together an introductory blog post, and then drove down to Fry's, picked up a bunch of LEDs, buzzers, and more, and lost the whole night to messing around with a Netduino and the .NET Micro Framework.

**DNC: Nice! You had a Bionic Arm project sometime back, what is it up to now-a-days?**
**PB:** My ADD (attention deficit disorder) really shows when it comes to projects. There are threads that flow throughout all my hobbies, but the specific projects tend to change quite a bit. The robotic ARM project was pretty neat, but then having done it, I lost interest in it. I still have a second robotic arm sitting in kit form in a box. I intend to give that to my son to do, probably in a year when he's around 8. Bionic ARM would be totally awesome. I was disappointed that on "Cyber Monday" here in the US, there were no bionic arms or brain implants for sale ;)

## "…what got you interested in electronics…"

## PB: "I blame Netduino!"

**DNC: Ahem! Thanks for the correction Pete, yes of course it's a Robotic arm not a Bionic arm, we are still a few years away from Bionics becoming hobby projects :). (Quickly) Moving on, give us a peek of 'Pete before Microsoft', work, interests, adventures and Dr. Who!**
**PB:** I spent well over a decade working for a consulting company, doing a variety of desktop apps, and some web apps. Lots of VB4-6, then Windows Forms, then WPF and finally, Silverlight. I was a Silverlight MVP for a couple years as well.

I've always been interested in creative toys (like LEGO), synthesizers, and yes, Dr. Who. (did you know there are LEGO-compatible Dr. Who sets?) Before joining Microsoft, I did a lot more woodworking as well, but then I had children and switched jobs, and the woodworking mostly stopped. My house is full of reminders that I need to take it back up again (unfinished cabinets and more). I did finally finish up the window bench seat I was building. I even managed to put doors on it (much to the relief of my wife who is still waiting for doors on the kitchen cabinets). Other than Dr. Who, I watch almost no TV, so spend a lot of time on other hobby projects and my family.

**DNC: We get a feeling 'multi-talented' doesn't quite begin to encompass your talents Pete. Coming to software development and Microsoft Technologies (XAML, WinRT and Silverlight), it's phenomenal how much work the 'Community' has done in the past on Silverlight and XAML ( referring to MVVM Light, Caliburn Micro, XAML Toolkit and host of other excellent projects) and yet Microsoft client side technologies are looked at as 'closed' or OSS hostile platform. Why do you think that is? What do you think people want when they say Windows Development should be more open/OSS Friendly?**
**PB:** I've always been excited about the different OSS projects that the community has contributed. I even use MVVM Light extensively in my Windows 8 XAML book.

As to more OSS on the desktop, this is a hard nut to crack. Many in the community say they want more OSS on the desktop, but then do not create it. Why don't they create it? Because, by far, most desktop developers are corporate developers, and using OSS in corporations is still much harder than it needs to be. Most companies just don't want to deal with it because of real or perceived licensing and support issues. In fact, many corporate desktop developers can't even use third-party controls – if it's not in the box with Visual Studio, they have to create it themselves. I know this both anecdotally as well as from personal experience at client sites before joining Microsoft.

Web development has a different pedigree. Its origins are in the OSS world, so there are fewer hurdles to adoption of OSS.

I think we'll see some of this turn around with Windows Store apps.

Developers there tend to be more OSS friendly and less corporate. Once the culture of developing using Windows Store apps starts off with using OSS, it sets the tone for the future. It's a good opportunity to turn the page on client-side OSS and make it the norm, not an exception.

**DNC: Two excellent points regarding brewing 'App Culture' and the relative ease of acceptance for OSS on the Web. This is also evident at Microsoft where the Web tooling team has had some success turning the boat to more OSS friendly water. As a developer (and not as a MS Evangelist), with WinRT now conceptually in place, do you think it's a good point where maybe WinRT development can become more transparent with public roadmaps and feature request updates etc. or do you think that's another couple of iterations away, before any thought is given to it? Maybe wait for it to mature a bit more?**
**PB:** That's really hard for me to say. WinRT has adopted some nice open standards (WinRT is itself COM + CIL metadata). As to whether or not it will be open, I know there are a lot of hurdles to having OSS components as part of your operating system.

**DNC: That segues us into the next question about the flip side of OSS - what are the key takeaways that a large company like Microsoft would look for when they embrace OSS for parts of their revenue generating software? It's easy to yell 'I want source code' what does it take to actually make that source code available?**
**PB:** I mentioned some hurdles above. Keep in mind that I am an OSS enthusiast, but not a legal expert by any means. I do know that licensing for OSS, while simpler on the surface, is considerably more complex when you're a large company with a lot of IP (both internally generated and licensed from third-parties) which threads its way through different products. It's easier to navigate those waters when you're looking at a new codebase, or something really obvious (jQuery being a good example). But, if you take something as simple as a client development stack for Windows – that may have licensed IP in it for working with audio/video, it may have other IP for file systems, and more. Most products at Microsoft make use of existing prior work as that enables us to deliver more, faster, and maintain compatibility. It's the Greenfield stuff which can more easily make use of (or license itself as) OSS. To be clear, we're encouraged to make use of and create OSS at Microsoft, we just need to be thoughtful and legally careful about how we do it.

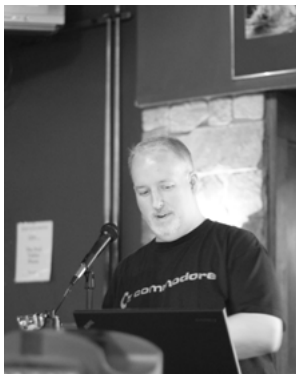
ASP.NET MVC gets a lot of kudos for being OSS, but I'd also like to point out the .NET Micro Framework. Not only is that fully OSS and Apache licensed, but the 4.2 version had something close to 50% of its code base community contributed (I don't recall the exact number, but it was between 40% and 60%). That's real OSS, with real impact on customers. This is for something which finds its way into machines on industrial shop floors, devices at hospitals, and more. I think that's a huge OSS success for us.

**DNC: That point about .NET Micro Framework needs a lot of highlighting! Fact is, Microsoft may-not appear to be a 'card carrying' member of the OSS club but they do have significant parts of their frameworks and tools either published as a standard or Open Source.**

# PETE ON WINDOWS 8 & SURFACE

**DNC: It's Windows 8 and Surface season, so we gotta have some questions here :). Have you adopted Windows 8 outside work? With the various low level I/O work that you do, did you experience any hiccups with Windows 8 or are you still saving a Windows 7 machine for those things?**
**PB:** My 6yo son has had Windows 8 running on his non-touch netbook since the consumer preview. I've been running it on various work machines since before the developer preview. I have two corp laptops which I use for presentations which run Windows 8 as well. One is touch, one is not. I'm getting my wife a new PC for Christmas, probably an Acer S7 running Windows 8.

My main PC at my house (the dual monitor 6 core beast I built in 2010) is running Windows 8 Pro. I have a lot of different types of things that have to run on this PC. It's where I do most of my writing. It has Cubase 6.5 and FL Studio for music production (as well as a ton of VST instrument plug-ins).
I have Messiah Studio for 3d animation, Adobe Creative Suite for image and media creation, Rhino 3d for CAD design. I have my MikroElektronika ARM compiler for board development and both Visual Studio 2010 and Visual Studio 2012 for other development. I have MIDI interfaces, pro audio interfaces, two 30" displays and more. I have two web cams (one so I can see over the screens to see when people come in my office), and a Kinect for Windows. I have three large powered USB hubs which are completely full. If anyone is a test case for whether Windows 8 will work on a complex setup, it's me :)



I also stood in line and purchased a 64 GB Surface for myself. As a Microsoft employee, I'll get a Surface in the future anyway, but I wanted one right then and for personal use. I absolutely love it. I love developing for it, and I love using it. I love the games on it, and I use it for reading books from my Kindle account. I've used it on the plane to write some of my Windows 8 book, and at the restaurant to keep the kids busy while waiting for food. I use it every night to read or play games a bit before going to bed. It replaced my Kindle Fire and some other things and has become the second device I carry everywhere (the first being my phone). It turns heads everywhere I bring it, including those of much younger generations. I've seen kids pointing at it and whispering to their parents at tables in restaurants more than once. It's great.

Ironically, the only Windows 7 machine I still have here is my corp-connected desktop PC. I'm waiting to see which brand-new Windows 8 device I get as my main corp machine before I take time to do anything with that. I primarily use that just for email and accessing internal web sites.

**DNC: That is truly awesome and speaks to the continuing tradition of Windows where backward compatibility has always been a key factor. Even with the Modern UI makeover, the fact that it still runs 'older applications' straight off the bat (on x86/x64 platforms) should be music to consumers' ears. To round off, what's your first impression of Surface RT? As a consumer, if you had to bet on it being a flier like the Xbox or a 'non-flier' like Zune where would your money be and why?**
**PB:** I love my Surface. Seriously, it's something that makes me really proud to work at Microsoft. The combinations of Windows RT plus our own hardware is just awesome.

I'm not a marketing guy, and am terrible about predicting the future. I think the strength of Surface as a tablet is in the apps. We already have a great story with Office for folks who want the productivity side. On the apps side (which my group has a lot of responsibility in shepherding), we're doing really well. The store is still new, but we already have tens of thousands of apps world-wide.

I had a Zune as well. Zune didn't get apps until much later so its differentiator was hardware and the catalog of music. The hardware we had in the final Zune generation was absolutely spectacular, but it seemed the public had already made up its mind by then. For the most part, public interest had moved to phones over dedicated music devices. Plus, Apple just had really effective marketing around the iPod and iPhone.

With Surface, our very first offering is extremely well done. I think by investing in the hardware and experience up front, we get to make a much better first impression. Add to that the marketing we've done, and the work we've done to get good apps into the Windows Store, and I'd say we're more on the Xbox side of the spectrum than the Zune side.

The other reason this works is because we have a very open hardware system. Third parties can create different form factors which appeal to different groups. You're not stuck with just a couple models that we create. If you don't like the Surface, but really like a Windows 8 or Windows RT tablet from someone else, you're free to choose that and still have the operating system fully supported and serviced from Microsoft.


Musician Jordan Rudess, with a Sony VAIO Tap 20, A Windows 8 PC in a rather unique form factor

As an aside, when I speak to musicians (one audience I've been working with is music app developers, folks like Image Line, Jordan Rudess/Wizdom, Open Labs, and many others), they are universally blown away by how we've managed to create a touch-friendly OS that scales from tablet to giant screens. The idea of playing on a really giant touch screen, the same app you purchased for your tablet, is something we have in our favor here. It helps validate the approach that it's a PC, regardless if it's a tablet or something sitting on your desk, and that most people need the flexibility of doing the same type of work no matter which machine they're at.

Let me leave you with this video I shot of Jordan when I set up his A720 at his house (featured video on my channel). It was his first time playing it, and his first time on a PC (he's a long time Mac and iOS guy).

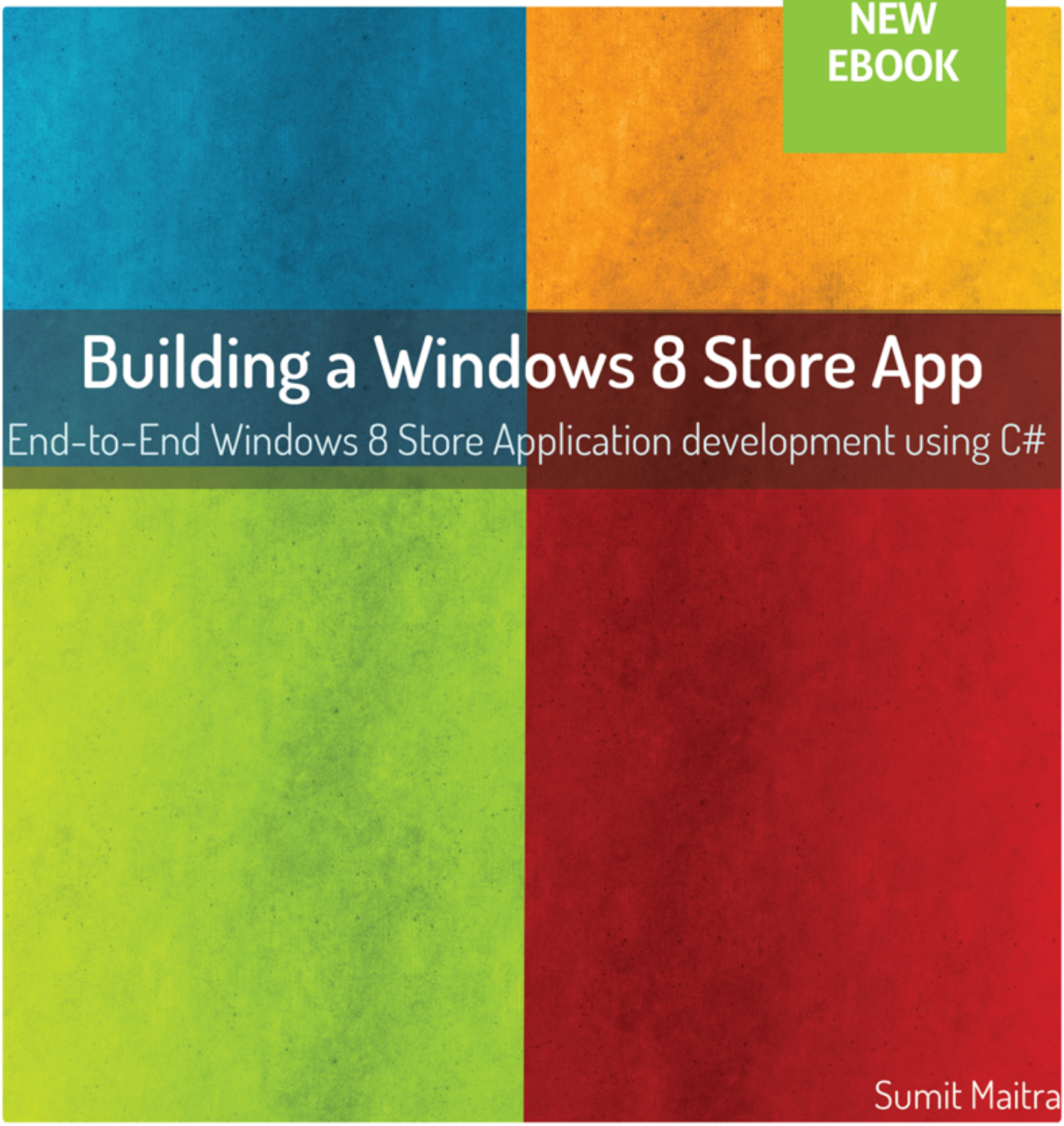
http://www.youtube.com/user/Psychlist1972?feature=mhee

**DNC: That's a nice video. We encourage our readers to view it and see how Jordan pretty much gets lost in his music with his hands on the giant Windows 8 PC. That, right there, is the 'immersive' experience the Windows 8 team has been talking about. We are pretty amped about the Surface ourselves and really want it to be Microsoft's next hit hardware!**

**With that we come to a close of this very nice 'conversation' here. Thanks again for your time Pete, wishing you and family a very happy Holiday season!**
**PB:** Thank you! And to you and your readers as well. ■

Planning to build apps for Windows 8?

NEW EBOOK




## Building a Windows 8 Store App

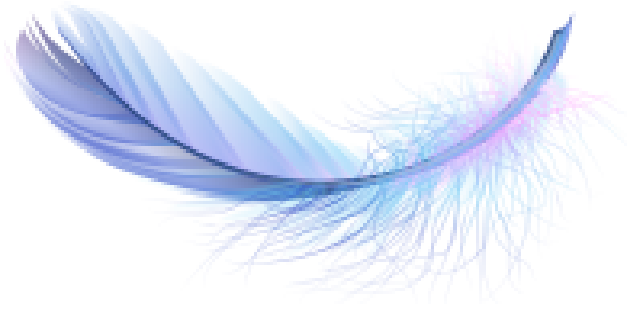
Are you interested in a book that shows how to create an End-to-End Windows 8 Store App using C# and XAML? Well we are writing a book to share the excitement and learning from the experience! Please click below to learn more.

**Click Here** www.windows8appsbook.com

# GETTING STARTED WITH MVVM LIGHT TOOLKIT

## FOR WINDOWS 8

PIOTR WALAT INTRODUCES THE MVVM LIGHT TOOLKIT FOR WINDOWS 8 STORE APPS AND WALKS US THROUGH THE FEATURES OF THIS POWERFUL LITTLE TOOLKIT.



MVVM Light Toolkit is an open source library created by Laurent Bugnion (@LBugnion) that aims to simplify Model-View-ViewModel pattern implementation in XAML applications. It is available for a wide variety of platforms including WPF, Silverlight, Windows Phone (7.x and 8) and Windows 8.

In this article, I am going to present some of the key project features and show how to use them to build a simple Windows Store app using C# and XAML.

## WHY MVVM?

Dealing with software complexity is one of the major challenges encountered in our industry. Complex projects are not only expensive to write but also (or even more importantly) to maintain, extend and test. Adhering to best practices and applying good design patterns can help companies reduce that cost significantly. This applies not only to backend code, but also to client applications, such as Windows Store apps. Model-View-ViewModel is an evolution of MVC pattern that lets

# WHY MVVM LIGHT?

*MVVM Light is arguably one of the most popular MVVM libraries out there, has decent documentation and is being actively developed.*

*One of the nice things about it is that it is not a framework, but rather a toolkit, a set of components aimed to help you apply the MVVM pattern. This means it does not enforce any coding conventions or constructs, but simply provides you with the tools you may freely choose from. Thanks to that approach, it is lightweight and additionally it is accessible to beginners due to its relatively flat learning curve.*

UI developers decouple application logic (such as data retrieval) from actual presentation description layer (such as XAML or HTML). It helps with software testability and maintainability.

MVVM is a great pattern that however comes at a certain cost – it requires 'boilerplate' code and can introduce potentially unnecessary abstractions. Before you start using the pattern, make sure that you really can benefit from it. If the project you are working on is small, has no unit tests and is unlikely to be extended in the future – you will be probably better off by keeping it simple and sticking with a code-behind backed approach. Having said that, other than demo code for blog articles, rarely are projects so small that they don't benefit from adoption of MVVM.

## GETTING STARTED

To start using the toolkit, simply download and run latest installer located at CodePlex (http://mvvmlight.codeplex.com/). The package not only contains essential binaries, but also bundles helpful code snippets and project templates (located in the installation folder) which is definitely a plus.

Please note that if you want to use MVVM Light in an existing project you can directly add the references from the Nuget packages available.

# FEATURES OVERVIEW

As mentioned before, MVVM Light Toolkit is basically a set of helper classes that enable programmers get up to speed when working on XAML MVVM projects. Below is a quick overview of some of these classes.

## OBSERVABLEOBJECT

ObservableObject is a base class implementing INotifyPropertyChanged (and INotifyPropertyChanging) interface. By deriving from it, you free yourself from a somewhat dull task of raising PropertyChanged event manually. One of the examples for its usage could be model classes that embody application domain entities. In this article, we will build a simple contact management application and for that purpose we want to introduce Contact class to represent a single person. Because we will display contact details on the screen, we want them to update whenever properties of the underlying object change.

```
public class Contact : ObservableObject
{
 public Guid Id { get; set; }
 private string _name;
```

```
 public string Name
 {
  get { return _name; }
  set
  {
   if(_name != value)
   {
    _name = value;
    RaisePropertyChanged(() => Name);
    //or alternatively
    //RaisePropertyChanged("Name");
   }
  }
 }
}
```

Please note that ObservableObject provides RaisePropertyChanged method overload that accepts either property selector expression (Expression<Func<T>>) or a property name string (which is the 'traditional' way of doing it).

If your data objects are generated by WCF client reference, they most certainly already implement INotifyPropertyChanged and you don't need to worry about it.

## VIEWMODELBASE

As its name implies, ViewModelBase is designed to be used as a base class for your view models. It derives from ObservableObject, which means that all INotifyPropertyChanged helper methods from this class are available. Moreover ViewModelBase implements ICleanup interface by providing Cleanup() method that can be overridden in deriving classes. The intent of ICleanup is to provide a consistent way of cleaning up after your view models (e.g. by flushing their state into a persistent storage).

ViewModelBase also contains other useful members, such as IsInDesignMode (and its static counterpart IsInDesignModeStatic) property that lets easily detect whether the view model has been instantiated in design mode (useful when using Blend).

## SIMPLEIOC AND VIEWMODELLOCATOR

Default MVVM Light template contains a class called ViewModelLocator that is designed to hold references to your view models. As with almost everything in the toolkit, you don't have to use this pattern if you decide that something else suits you better, however it helps to abstract actual view model instantiation and provides a good place to inject any view model dependencies.

At this point, it is worth mentioning that the toolkit comes with a very simple IoC container – SimpleIoc. It implements the IServiceLocation interface provided by .NET Patterns and Practices extensions. SimpleIoC can be used as follows to hook up Interfaces to their concrete implementations as well as register Types in the type cache.

```
static ViewModelLocator()
{
 ServiceLocator.SetLocatorProvider(() => SimpleIoc.
Default);
 SimpleIoc.Default.Register<IDataService, DataService>();
 SimpleIoc.Default.Register<MainViewModel>();
 SimpleIoc.Default.Register<SecondViewModel>();
}
```

As you can see, its usage is very straightforward. In the example above, we register DataService as IDataService implementation. If any of the two view models depend on IDataService, this dependency will be autowired.

```
public class SecondViewModel : ViewModelBase
{
 private readonly IDataService _dataService;
 //dataService will be injected by the container
 public SecondViewModel(IDataService dataService)
 {
  _dataService = dataService;
 }
    //(...)
}
```

## RELAYCOMMAND

Because .NET events don't really work well with MVVM pattern (they are not bindable), XAML frameworks usually use commands to react to user actions. Simply put commands are classes that implement ICommand interface.

```
public interface ICommand
{
 event EventHandler CanExecuteChanged;
 bool CanExecute(object parameter);
 void Execute(object parameter);
}
```

WinRT (just like Silverlight) does not ship with any ICommand implementations out of the box and leaves this task to the developer. Luckily MVVM Light provides RelayCommand and RelayCommand<T> classes that help you save some time.

Note that only Button based controls accept an ICommand instance and attach it to the Click or Tap event.

If you are familiar with MVVM Light on other platforms, you may have used EventToCommand behavior which lets you use events as commands easily. Unfortunately it is unavailable in Windows 8 version of the toolkit - although there are other third party implementations you can use.

# CREATING A SIMPLE MVVM APPLICATION

Let's create a simple MVVM application to show how to use toolkit's features in practice. The application will consist of one page only and will let user add, delete and display contacts.

## ICONTACTSERVICE

Because we don't want to pollute our view with data retrieval logic, we are going to abstract this functionality into a separate component - IContactService.

```
public interface IContactService
{
 Task<IEnumerable<Contact>> GetAllAsync();
 Task AddAsync(Contact contact);
 Task DeleteAsync(Guid id);
 Task UpdateAsync(Contact contact);
}
```

Communication over network is usually a long-running process (from UI perspective that is), thus we want to make it asynchronous and ideally use async/await pattern.

To keep our example simple, we will provide a mock implementation that uses in-memory store instead of actually calling any web services. In real world, you probably will want to use HttpClient or a generated web service proxy.

```
public class ContactService : IContactService
{
 readonly IRepository<Contact> _repository =
   new InMemoryRepository<Contact>();

 public async Task<IEnumerable<Contact>> GetAllAsync()
 {
  //introduce a delay to simulate long running process
  await Task.Delay(250);
  return _repository.Items.ToList();
 }

 public async Task AddAsync(Contact contact)
 {
  await Task.Delay(250);
  _repository.Add(contact);
 }

 public async Task DeleteAsync(Guid id)
  {
  await Task.Delay(250);
    _repository.Delete(id);
  }

  public async Task UpdateAsync(Contact contact)
  {
   await Task.Delay(250);
   _repository.Update(contact);
  }
 }
}
```

## VIEW MODEL

Let's create our view model now. Apart from other things, it will be responsible for loading contacts and holding a reference to currently selected item.

```
public class MainViewModel : ViewModelBase, ILoadable
{
 private readonly IContactService _contactService;
 private Contact _selectedContact;
 private List<Contact> _contacts;

 public List<Contact> Contacts
 {
  get { return _contacts; }
  set
  {
   if(_contacts != value)
   {
    _contacts = value;
    RaisePropertyChanged("Contacts");
   }
  }
 }

 public Contact SelectedContact
 {
  get { return _selectedContact; }
  set
  {
   if (_selectedContact != value)
   {
    _selectedContact = value;
    RaisePropertyChanged("SelectedContact");
   }
  }
 }
```

```
 public MainViewModel(IContactService contactService)
 {
  _contactService = contactService;
 }
 public async Task LoadAsync()
 {
  var contacts = await _contactService.GetAllAsync();
  Contacts = new List<Contact>(contacts);
 }
}
```

ILoadable is a very simple interface used to mark view model as loadable at startup.

```
public interface ILoadable
{
 Task LoadAsync();
}
```

## VIEWMODELLOCATOR AND DESIGN-TIME DATA

Having core view model functionality in place, let's continue by creating ViewModelLocator class. This example shows how easy it is to use SimpleIoc class for dependency injection. Please note that we want to be able to see mock design-time data and that is why we provide a special IContractService implementation (DesignContactService) that will be responsible for providing contacts in Blend (or Visual Studio designer).

```
public class ViewModelLocator
{
 static ViewModelLocator()
 {
  ServiceLocator.SetLocatorProvider(()
    => SimpleIoc.Default);
  if (ViewModelBase.IsInDesignModeStatic)
  {
   SimpleIoc.Default.Register<IContactService,
    DesignContactService>();
  }
  else
  {
    SimpleIoc.Default.Register<IContactService,
     ContactService>();
  }
  SimpleIoc.Default.Register<MainViewModel>();
 }
 public MainViewModel Main
 {
  get
  {
   return ServiceLocator.Current.
GetInstance<MainViewModel>();
  }
 }
}
```

The locator can be instantiated as a static resource in App.xaml.

```
<Application.Resources>
 <ResourceDictionary>
 <!--Global View Model Locator-->
  <vm:ViewModelLocator x:Key="Locator"
    d:IsDataSource="True" />
 </ResourceDictionary>
</Application.Resources>
```

The view itself is going to be simple and will consist of a single ListView (using WrapPanel as ItemsPanel) and an AppBar providing buttons for refresh, add and delete actions. These buttons will be bound to view model commands (we will commands later on).

```
<ListView Grid.Row="1" SelectionMode="Single"
  ItemsSource="{Binding Contacts}"
  SelectedItem="{Binding SelectedContact, Mode=TwoWay}"
Margin="80,50,80,50">

<Page.BottomAppBar>
 <AppBar IsOpen="True">
  <Grid>
   <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
   </Grid.ColumnDefinitions>
   <StackPanel Orientation="Horizontal">
    <Button Style="{StaticResource
RefreshAppBarButtonStyle}"
     AutomationProperties.Name="Refresh Data"
     Command="{Binding RefreshCommand}"/>
    <Button Style="{StaticResource AddAppBarButtonStyle}"
     AutomationProperties.Name="Add Item"
     Command="{Binding AddCommand}"/>
   </StackPanel>
   <StackPanel Grid.Column="1" HorizontalAlignment="Right"
     Orientation="Horizontal">
   </StackPanel>
  </Grid>
 </AppBar>
</Page.BottomAppBar>
```

We also need to set DataContext of our page as shown below.

```
<common:LayoutAwarePage x:Class="MvvmLight.MainPage"
   DataContext="{Binding Main,
   Source={StaticResource Locator}}">
```

In order to benefit from design-time data support (this helps immensely when designing XAML views in Blend), we need to do one more thing. In our view model's constructor, if design mode

has been detected, we need to explicitly load the data from IContactService.

```
public class MainViewModel : ViewModelBase, ILoadable
{
 public MainViewModel(IContactService contactService)
 {
  _contactService = contactService;
  CreateDesignTimeData();
 }

 [Conditional("DEBUG")]
 private async void CreateDesignTimeData()
 {
  if (IsInDesignMode)
  {
   await LoadAsync();
  }
 }
}
```

Please note, that CreateDesignTimeData has been decorated with ConditionalAttribute, which means it will get executed only if DEBUG symbol has been defined. Because we will be executing this code when in design mode, IoC container will inject the view model with DesignContactService instead of ContactService

From now on we should be able to benefit from design-time data in Blend and Visual Studio XAML designer.

## ADDING COMMANDS TO VIEW MODEL

Let's add missing commands to MainViewModel using RelayCommand class provided by MVVM Light Toolkit.

```
public RelayCommand AddCommand { get; set; }
public RelayCommand RefreshCommand { get; set; }
public RelayCommand DeleteCommand { get; set; }
public RelayCommand UpdateCommand { get; set; }

public MainViewModel(IContactService contactService
    , ICreateContacts contactCreator)
{
 _contactService = contactService;
 _contactCreator = contactCreator;
 CreateDesignTimeData();
 CreateCommands(); }

private void CreateCommands()
{
 AddCommand = new RelayCommand(AddHandler);
 RefreshCommand = new RelayCommand(RefreshHandler);
 DeleteCommand = new RelayCommand(DeleteHandler,
     () => SelectedContact != null);
}

private async void DeleteHandler()
{
 IsBusy = true;
 await _contactService.DeleteAsync(SelectedContact.Id);
 IsBusy = false;
 await LoadAsync();
}

private async void RefreshHandler()
{
 IsBusy = true;
  await LoadAsync();
 IsBusy = false;
}
```

CanExecute predicate passed in as an argument of DeleteCommand constructor will automatically disable the command if no contact has been selected. In order for this functionality to work properly, we need to invoke DeleteCommand.RaiseCanExecuteChanged() every time selected item changes.

```
public Contact SelectedContact
{
 get { return _selectedContact; }
 set  {
  if (_selectedContact != value)
  {
    _selectedContact = value;
     RaisePropertyChanged("SelectedContact");
     DeleteCommand.RaiseCanExecuteChanged();
  }
 }
}
```

Adding a contact will most likely involve navigating to a separate page or presenting user with a dialog. Because it is presentation logic it should not be incorporated directly into the view model. The view model does not need to know who or how can new contacts be created, as all it really cares about is getting a new instance of Contact class. To achieve loose coupling, we can do one of two things: use Messenger helper class or abstract contact creation logic into a separate component (ICreateContacts) that would be injected by IoC container into the view model.

For the sake of simplicity, we are going to use second approach and create a trivial ICreateContacts implementation.

```
public class DummyContactCreator : ICreateContacts
{
```

```
 public async Task<Contact> CreateContact()
 {
  var contact = new Contact()
  {
   Name = “Jane Doe”,
   Email = “jane.doe@someservice.com”,
   TwitterHandle = “@janestwitter23”
  };
  return contact;
 }
}
```

The last thing left to do is to register this class in our IoC container. SimpleIoc.Default.Register<ICreateContacts, DummyContactCreator>(); After this step, our example should be functional.

At this point if we run the application, we can add Contacts to our Contacts App by clicking on the ‘Add’ button from the Task Bar that appears on right-click or ‘scroll-up-from-the-bottom-edge-of-the-screen’ gesture.

## SUMMARY

MVVM Light Toolkit is a wonderful library that can make MVVM XAML development much easier if used correctly. Recently Windows 8 support has been added to the toolkit, allowing Windows Store app developers to use it in their projects. It is strongly recommend that the MVVM pattern adopted for any serious XAML project. MVVM Light provides a nice set of tools to help us do so, it is simple to use, works on many XAML platforms including Windows Phone and is being actively developed.

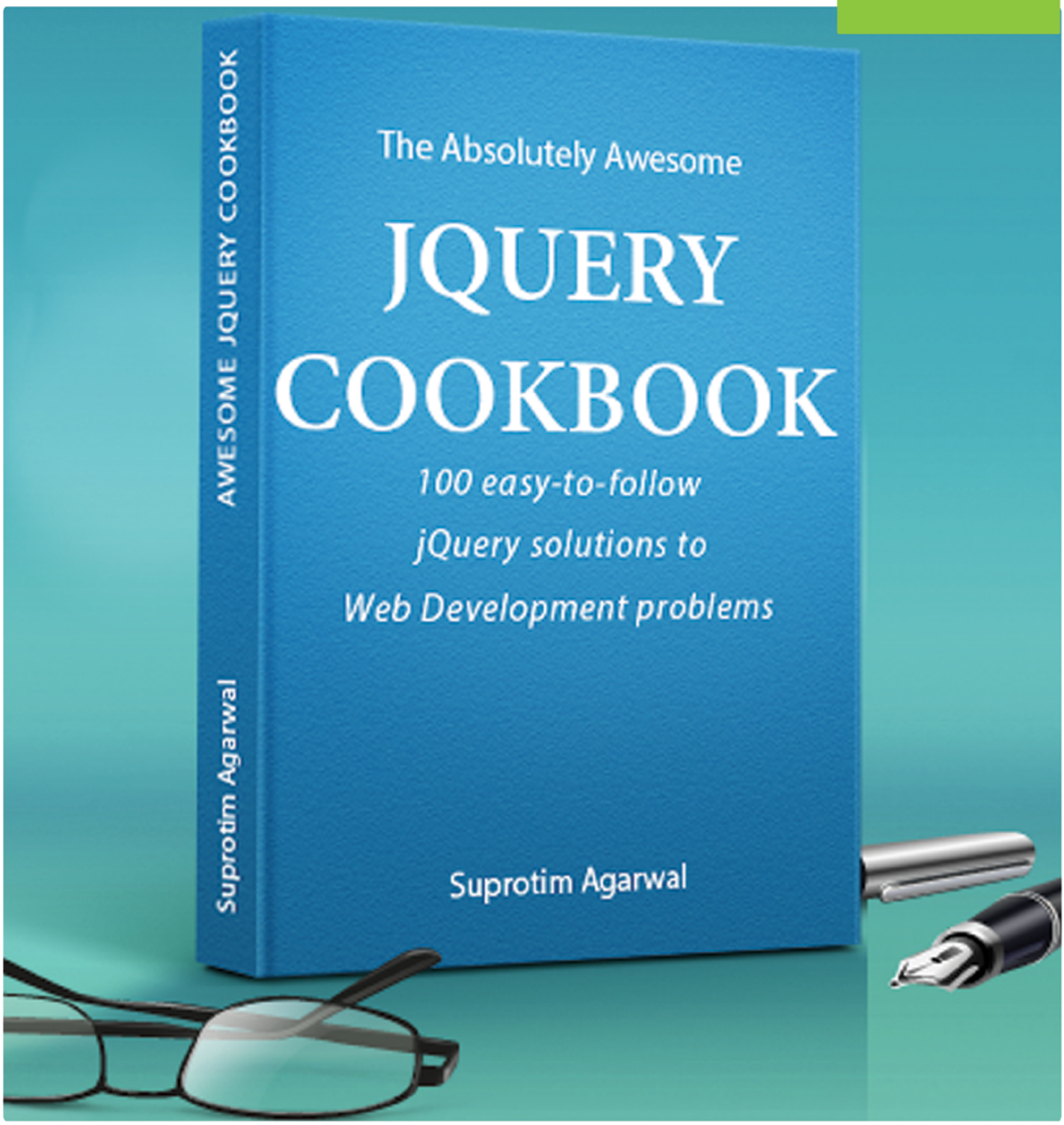
Make sure to have a look at pretty impressive set of examples and tutorials available at http://www.galasoft.ch/mvvm/#tutorials. If you are on Twitter follow @LBugnion for the latest updates on MVVM Light. ■

Source Code for the sample can be downloaded at http://bit.ly/dncm-04-mlc

Piotr Walat is a senior software developer using C# as his primary language. Currently he works for a digital finance company in Dublin, Ireland. In the past he had the pleasure of working for big IT names such as AOL and Microsoft. His interests include modern software architectures, web and mobile application development as well as artificial intelligence. In his free time, Piotr enjoys reading books and playing soccer. You can visit his blog at www.piotrwalat.net or catch up with him on Twitter @pwala

# The Absolutely Awesome jQuery Cookbook

NEW EBOOK




## 100 Easy-to-follow jQuery solutions

With scores of practical jQuery recipes you can use in your projects right away, this cookbook will help you gain hands-on experience with the jQuery API!

Please click below to learn more.

**Click Here** www.jquerycookbook.com

# REAL WORLD OAUTH

## USING ASP.NET MVC 4

**Sumit Maitra** shows how to hack into the new OAuth integration using DotNetOpenAuth in an ASP.NET MVC application and retrieve all possible user information from the Provider.

If you haven't heard of OAuth or OpenID, you have most certainly used it in day to day life. Have you provided your google username/password and logged in to StackOverflow or used your WordPress.com account to post comments on someone's blog or used third party Twitter clients whom you have given permission to use your Twitter account or used Facebook to post comments on some tech blog site? If answer to any of the above is a yes, you have seen OAuth or OpenID in action.

### WHAT IS OPENID AND OAUTH

At a 100K feet both are Authentication protocols that work towards a dream on the web, that is - Single Sign On. Basic premise being, instead of having to remember a different username and password for every Application we visit on the internet, we have a dedicated Authentication provider with whom we register our user name and password. Applications (aka Relying Party) redirect us to this provider for authentication and after a successful authentication, the provider passes back a token to the Application who initiated the request. The target Application then uses the token to verify the identity of the person and provide further access to Application Features.

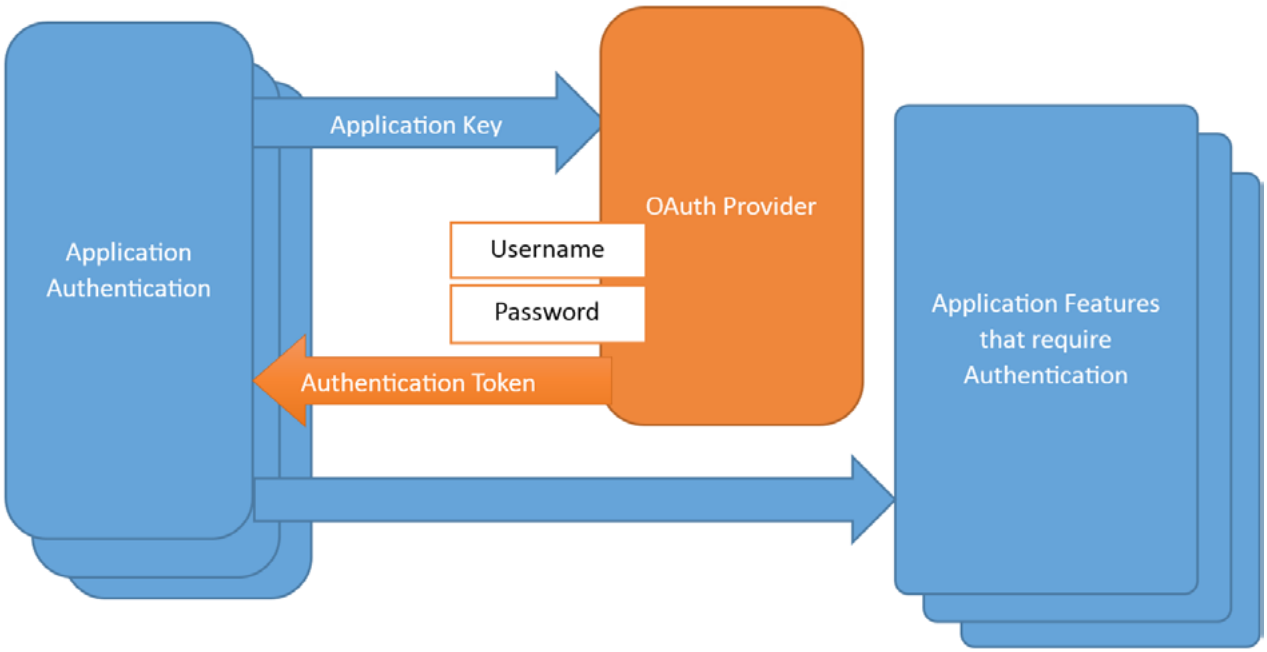
OAuth was initiated by Twitter when they were working on building their OpenID implementation for Twitter API. Their need was more towards controlling what features to make available based on Authentication Types offered at the Provider and those sought by the relying party.

For example when a Relying party signs up for Twitter Authentication access, they can request for a Read-Only connection (this gives them read permissions to a user's stream), a Read/Write connection (this gives them read and write permissions on a user's stream) and a Read/Write with Direct Messages connection (giving them maximum possible access to user's data).

Thus, when a user authenticates with Twitter over OAuth, they are told exactly what kind of access they are providing.

On the other hand, OpenID simply ensures that you are who you claim to be by verifying your username and password. It has no way of controlling feature access.

In even simpler terms, OpenID is about Authentication, OAuth is about Authentication and Authorization (for features at the Provider).




The diagram above demonstrates the Authentication process for an OAuth provider. The overall workflow is as follows:

1. Application (Relying Party) registers with OAuth provider and obtains an Application specific key (secret). This is a one-time process done offline (not shown in above image).
2. Next the Application (Relying Party) has to pass the Application key to the provider every time it intends to authenticate someone.
3. User of the Relying Party needs to be registered with the Provider (again a one-time offline process)
4. Every authentication request directs the user to the Auth Provider's site. The user enters the Username and Password that they obtained in Step 3 above. The Provider verifies the account credentials, then it matches the Application Key and provides a token back to the Relying Party with which only the Authorized actions can be performed.

I have a bunch of references up on the Github page for this article. Feel free to go through them if you would like to understand these protocols in greater depth.

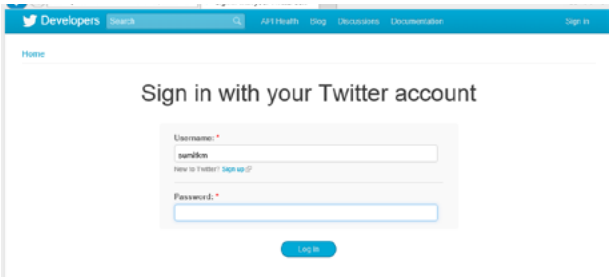
## TWITTER OAUTH AND ASP.NET

Today we will learn how to use ASP.NET's integration of the DotNetOpenAuth library that allows you to integrate using Twitter/Facebook/Google Id and other similar providers. Most examples on the net kind of stop after the Authentication piece and don't elaborate on how you can use it further. Today we'll dig a little deeper and see what it takes to go beyond the default implementation provided by ASP.NET and extract further information from our OAuth provider i.e. Twitter.

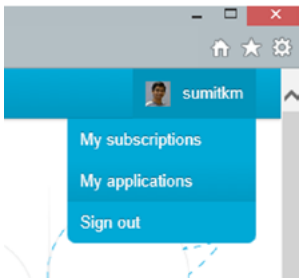
### Getting Started – Registering an Application with Twitter

As we saw above, there are four steps. Let's get started with Step 1, registering our application with Twitter.

1. Navigate to http://dev.twitter.com/ and click on the Sign In (top right)

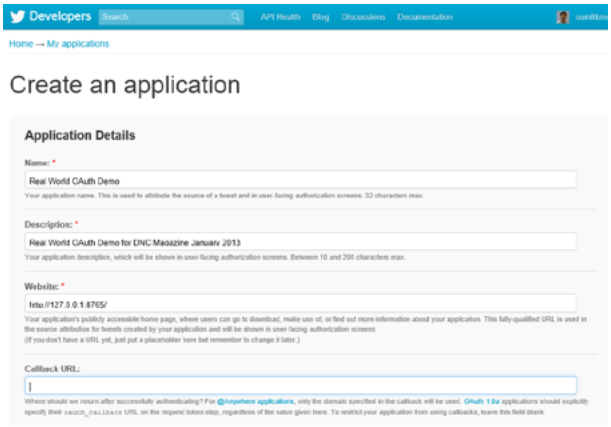



2. Log in using your Twitter credentials. Hover over your Account Info and Click on 'My Applications'

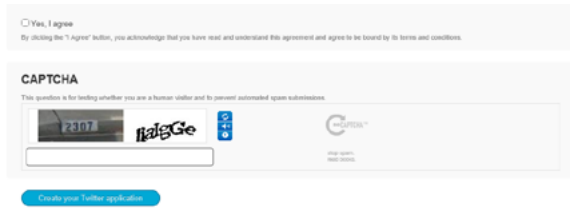


This will take you to the Applications page where you can register a new Twitter Application. This is going to be the Relying Party.

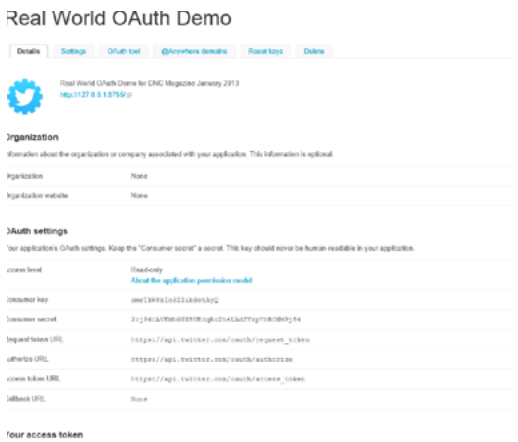
3. Next click on 'Create a new application' and navigate to the new application page. It asks for the following details




• Provide proper Name and Description these will come up every time a new user Authenticates at your site.
• Specify a callback URL that would point to a site address. You can point it to http://127.0.0.1/
• Notice the Website is setup as localhost address. This is for testing purposes. Once you have the application going set it back to the actual website.
• Scroll down, read the Developer Rules of Road carefully and click on Yes, I agree. Fill in the captcha and click on 'Create your Twitter Application'.




4. Twitter will navigate to a page similar to the following:




Next click on the 'Create my access token' to generate the access token that our app will be using along with the Consumer Key and Consumer secret. Remember all three values shouldn't be shared.

Also notice the Access Level is set to Read-only by default. For our application we need Read-only access, but if you would like to post tweets on the User's behalf you can modify the permissions from the 'Settings' tab. Remember to regenerate the Access Token if you change the Access Level. With that we are ready with Step 1. Let's build the barebones of our Application now.

## Using Registration information to Authenticate with Twitter

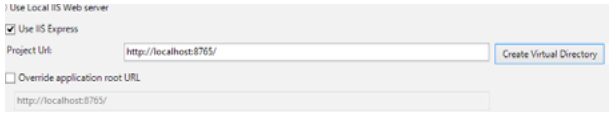
1. Create a new ASP.NET MVC Web Application and select the Internet Template.

2. Open the AuthConfig.cs from the App_Start folder and uncomment the code to register the Twitter Client

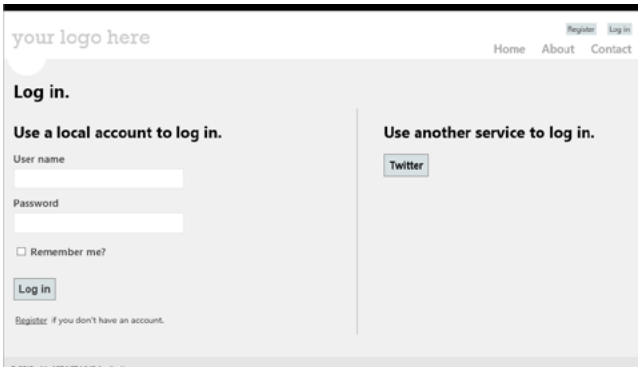
```
public static void RegisterAuth()
{
 OAuthWebSecurity.RegisterTwitterClient(
  consumerKey: "",
  consumerSecret: "");
}
```

3. Put the keys in web.config and update the above code to get the values from ConfigurationSettings.

4. Update the project's properties to run the application on the same port as specified in the URL above

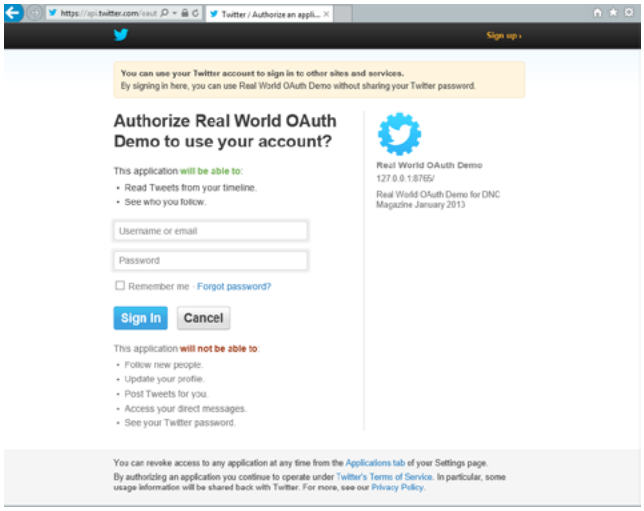



5. That's pretty much it. Run the Application and click on the Log On Link to be presented with the new Login page




As you can see we have a button to log in to Twitter. Let see what happens when we click on it.

6. On clicking twitter the application gets routed to a Twitter page




Here you will notice the following

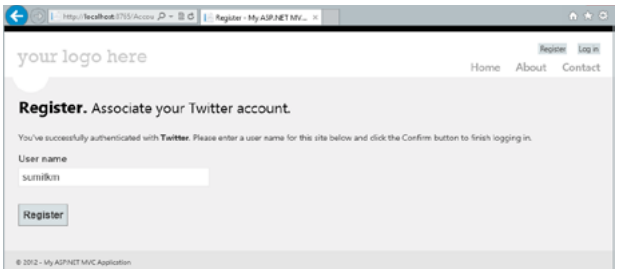
- On the right hand side is information about your app that you provided while creating it, including the URL.

- Above the Username or email address text box is a list of things that the requesting application can do, in our case it can 'Read Tweets from your timeline' and 'See who you follow'.

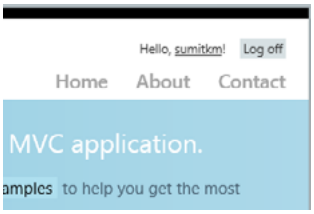
Note: This is why when using OAuth you have to be careful to request for only as much information as you need. If your app seems terribly nosy, the end user may choose not to give you permissions.

- On successful authentication, you will see a brief flash of 'redirection in progress' message and the user will be sent back to our application again.

7. For the first time log in, we will be requested to register via the following page



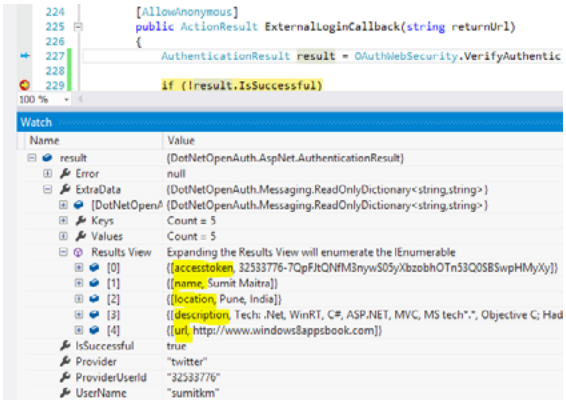

8. Once you Register, you are logged in as an Authenticated user for the application




9. With this we have completed Step 2, 3 and 4 of the OAuth process and this is where most demo applications draw a line.

## HACKING INTO THE OAUTH CLIENT

Since we have a working Twitter Authentication going, let us see what it takes to use the Twitter Avatar in our application. Once authenticated, we expect the entire User profile to be returned to us. This profile in the Authentication result as ExtraData. However if we break in the AccountController soon after the log in, we will see ExtraData only has a handful of values that does not include the Avatar URL.




What now? After some head-scratching, I concluded that there was no other option but to create a custom OAuth client for ourselves to get the required data. Essentially the default TwitterClient class that the ASP.NET team is provided is barebones to say the least.

### Overriding the default TwitterClient to build one of our own

To start off with, I've referred to the TwitterClient class from DotNetOpenAuth's Github repository (yay for OSS), and copied the entire implementation over into a new class called TwitterClientEx.

After copying the contents over, we will need to copy over some helper methods for formatting and escaping characters, these are mysteriously implemented as internal methods.

The code in context refers to the internal static extension class DictionaryExtensions that has some helper methods. We also bring in three helper methods LoadXDocument FromStream, CreateUntrustedXmlReaderSettings and EscapeUriDataStringRfc3986.

All these methods are implemented in the TwitterClient or OAuthClient classes but are unfortunately not reusable.

With the above in place, we can register it in AuthConfig.cs using the following code.

```
OAuthWebSecurity.RegisterClient(new TwitterClientEx(
    consumerKey: ConfigurationManager.
AppSettings[“consumerKey”],
    consumerSecret: ConfigurationManager.
AppSettings[“consumerSecret”]), “Twitter”, null);

//OAuthWebSecurity.RegisterTwitterClient(
//    consumerKey: ConfigurationManager.
AppSettings[“consumerKey”],
//    consumerSecret: ConfigurationManager.
AppSettings[“consumerSecret”]);
```

As we can see above, we have commented out the RegisterTwitterClient static method and instead used the generic RegisterClient method and passed in our own TwitterClientEx instance. The second parameter is simply the Label by which the Client is referred to.

However so far we have merely copied the TwitterClient so we still don't have our Avatar URL. To load the Avatar and other User Data let's revisit the VerifyAuthenticationCore method in TwitterClientEx.

The default code has only the following parameters extracted from the XML Document returned by Twitter.

```
extraData.AddDataIfNotEmpty(document, “name”);
extraData.AddDataIfNotEmpty(document, “location”);
extraData.AddDataIfNotEmpty(document, “description”);
extraData.AddDataIfNotEmpty(document, “url”);
```

However, instead of hardcoding as shown above, if we have the following Loop, then we can extract the complete set of values in the ExtraData dictionary.

```
XDocument document = LoadXDocumentFromStream(respons
eStream);
foreach (var node in document.Descendants(“user”).
Nodes<XElement>())
{
 if (node.NodeType == XmlNodeType.Element)
 {
  extraData.AddDataIfNotEmpty(document,
((XElement)(node)).Name.LocalName);
 }
}
```

Now that we have retrieved the data required, let's see how we can save it for ourselves.

## Saving 'ExtraData'

By default, the ViewModel objects and Data Model objects are bunched together in Model\UserAccounts.cs file that has multiple class files. Even though I dislike multiple classes per file, I'll ignore it for the moment and we'll simply extract the UserProfile class to a new Model\Account folder.

In this folder, we will add our ExtraData class. The POCO looks as follows

```
public class ExtraData
{
 public int Id { get; set; }
 public string AccessToken { get; set; }
 public string Name { get; set; }
 public string Location { get; set; }
 public string Description { get; set; }
 public string Url { get; set; }
 public UserProfile ParentUserProfile { get; set; }
 public int UserProfileUserId { get; set; }
}
```

For brevity, we will refrain from saving all the 35 possible values that can be returned in ExtraData resulting from the Authentication.

The POCO has a parent-child relationship established with the UserProfile object.

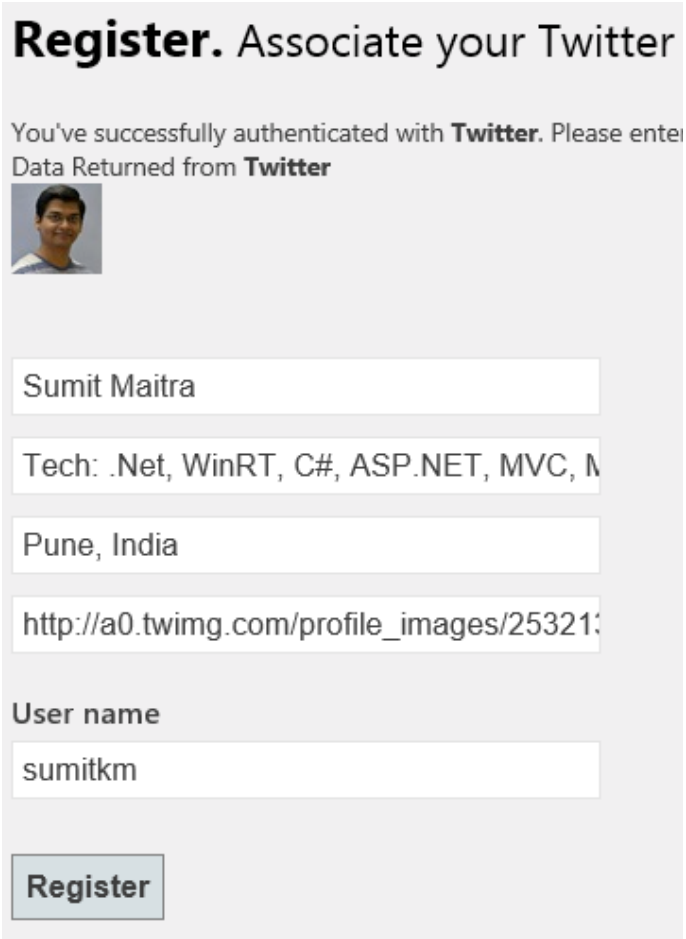
In the AccountController, the ExternalLoginCallback method is called after user returns from Twitter's Authentication page. Here the result variable retrieves all the information returned after a successful Authentication. We populate our ExtraData POCO from the result here.

As a next step, the boilerplate checks if the user exists locally and for first time, access the code redirects user to a confirmation page and requests for UserId to be used in this application. This is where we can update the ExtraData poco into the database. However we need to pass all the data via the RegisterExternalLoginModel to the ExternalLoginConfirmation page. To do this, we update the RegisterExternalLoginModel to carry ExtraData with it.

```
public class RegisterExternalLoginModel
{
 [Required]
```

```
 [Display(Name = "User name")]
 public string UserName { get; set; }
 public string ExternalLoginData { get; set; }
 public ExtraData AuthenticationProviderData { get; set;
}
}
```

In the View, we add a set of Labels and Hidden Variables and an Image to represent the ExtraData on the View. There we have it, Authentication with Avataar.




Once we click Register, the ExtraData is saved in the Application's database and we can retrieve and use it anytime we want.

## EXAMPLE USAGE

Now that we have seen how to Authenticate against Twitter or other OAuth providers, we can use this technique for creating a Disqus like commenting system or a StackOverflow style Forum or any 'Socially connected' site we want.

The default implementation that comes out of the box with ASP.NET is a little light for direct practical usage, but as we saw, it's not really difficult to extend it to our benefit.

## CONCLUSION

We saw how we can build our ASP.NET application to use Twitter for Authentication. The usage scenario discussed was only the tip of the iceberg. DotNetOpenAuth is a powerful library and hidden under the lightweight wrappers created by the ASP.NET team, is a real powerhouse framework that can take care of a lot of your Social Media integration requirements. ■

The complete code can be downloaded from Github at http://bit.ly/dncm-04-oaut

Sumit is a .NET consultant and has been working on Microsoft Technologies for the past 12 years. He edits, he codes and he manages content when at work. C# is his first love, but he is often seen flirting with Java and Objective C. You can Follow him on twitter @sumitkm and read his articles at bit.ly/KZ8Zxb

# CUSTOMER FIRST: STORYBOARDING & CLIENT FEEDBACK USING TFS 2012

ALM Expert and Microsoft MVP **Subodh Sohoni** walks us through a possible workflow using Visual Studio TFS 2012, specifically Storyboarding and Feedback Client tools. He demonstrates a scenario that can help drive iterative software development while keeping Customers in the loop all the time.

In this article, we will look at how TFS 2012 can help implement an Agile process keeping the focus on the Customer or Customer Representative. We will look at, not just Agile principles, but also how we can empower the Customer Representative, to maintain agility in projects that have multi-site development teams.

To start off with, let's outline the points we want to keep in mind as we go about establishing the process:

*Highest priority is to satisfy the customer through early and continuous delivery of valuable software.*

*Welcome requirement changes even late in development cycle. Agile processes are meant to enable continuous improvements for customer's competitive advantage.*

*Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for shorter turnarounds.*

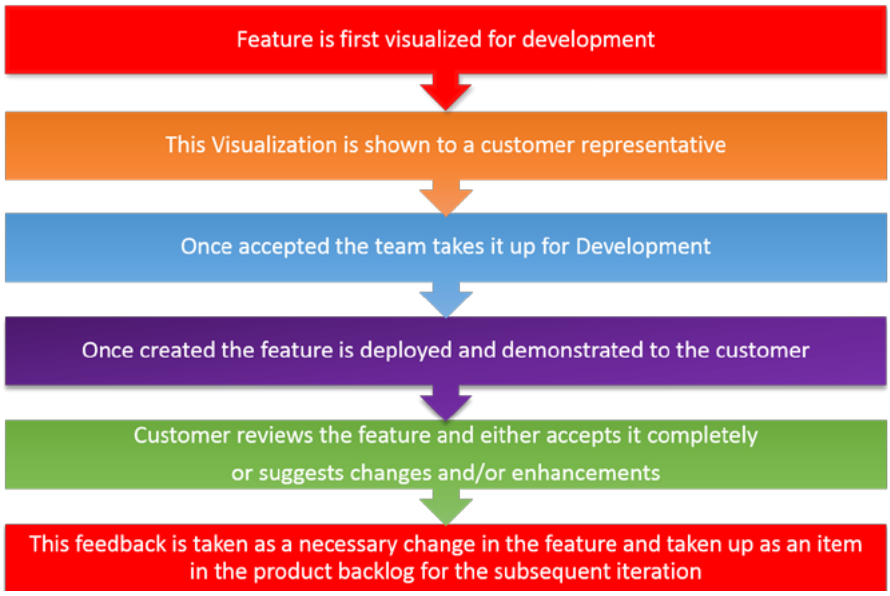
*Business and Developers must work together daily throughout the project.*

*Working software is the primary measure of progress.*

As it becomes obvious, if an organization wants to adhere to these principles, it needs to follow an iterative process.

During an iteration, a subset of the features scoped in the entire product backlog, should be implemented. If the team wants to involve the customer to work together daily, then customer representative should be aware of what the team is developing and give feedback about it before and after it is developed.

A possible prototype for the Process could look something like this –




## A COMMON CHALLENGE

A question always comes to mind regarding implementation of the above process. How is it possible to implement it in a scenario where the customer is not immediately available for interactions? One of the main reasons may be that the location of development team is away from the customer's location. The important aspect of Agile - Customer consideration in building up of the software, becomes a suspect.

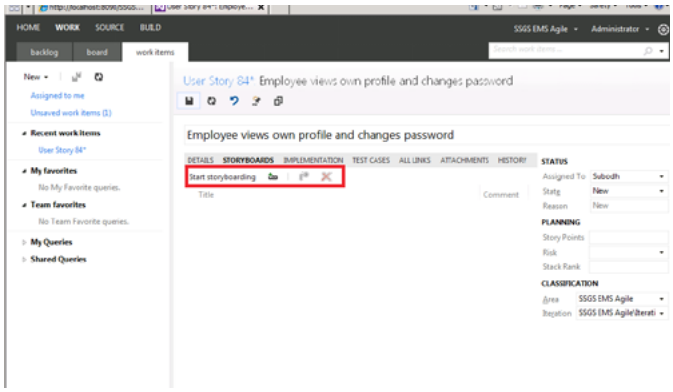
## LEVERAGING VISUAL STUDIO TEAM FOUNDATION SERVER 2012

In Visual Studio Team Foundation Server 2012, Microsoft has answered the above concern quite effectively. It has provided two features - Storyboarding and Feedback Client that integrates with TFS 2012. Storyboarding uses a commonly used component of MS Office, PowerPoint! It provides a way to create a storyboard that represents a user story and links it to appropriate work item. Feedback Client is a standalone application that can be used to get the feedback about the deployed software from the customer. Let us go into details of each one of them.

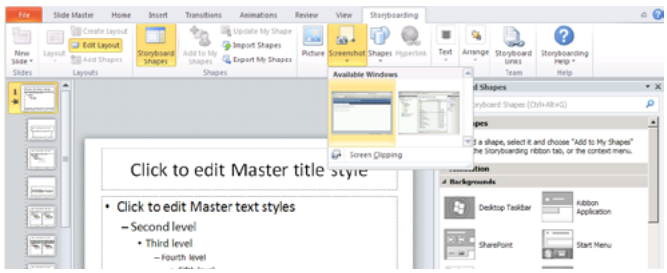
## STORYBOARDING

Storyboarding usually starts when a User Story has been identified and the team needs to get approval for its flow from customer representative. Team begins by creating a User Story / PBI / Requirement work item. Team has some idea about the story but needs to get confirmation of their perception from the Customer Representative.

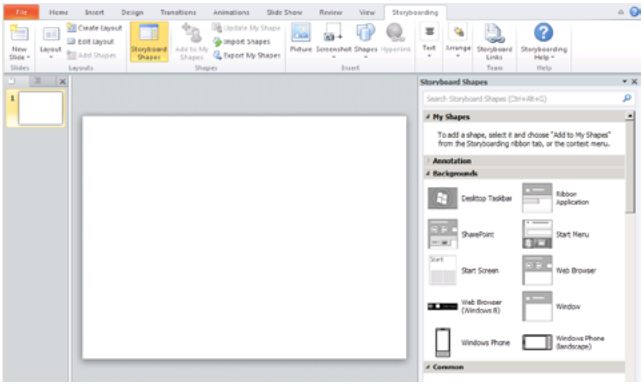
So now the team creates a storyboard of their perception. When that work item is opened on a machine that has Visual Studio 2012 and MS Office (2007 onwards) installed, it is possible to create a linked storyboard. Work item form has a tab for STORYBOARDS. On that tab, the menu item allows us to 'start storyboarding'. When it is clicked, it starts PowerPoint and creates a new presentation in it. It is also possible to create a storyboard first in PowerPoint and then link it back to an existing work item.



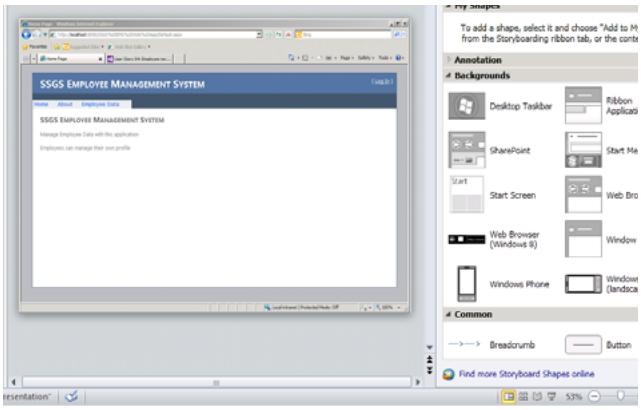

The storyboarding capability of PowerPoint includes a ribbon to manage the following:



1. Layout that has look and feel that is similar to actual software being developed. We can export created layouts and import them back on other machines.



2. If UI of the software already exists, then we can take a screenshot of that as background of layout. This comes in handy for UI Enhancement type of changes where the existing Layout forms the background and the changes can be highlighted using new components.

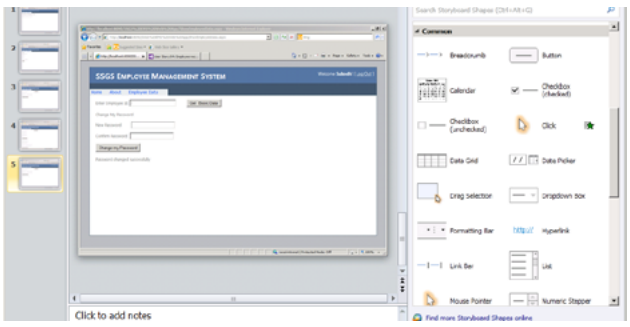


3. Shapes that can help us create the mock of actual user interface of the software. There are a few standard shapes that are provided but we may create some composite shapes and export them for others to use.

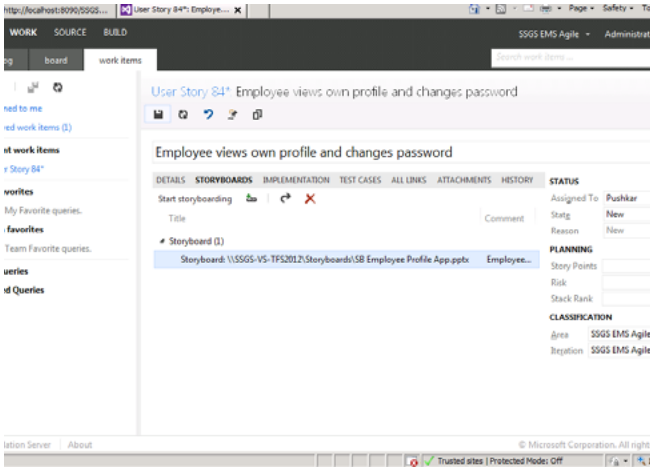
4. Graphics manager that allows text and shapes alignment etc.

5. Links to the work items.

6. Shapes for various types of applications are included by default. We can use those to create storyboards of web applications, SharePoint applications, Windows Phone apps or desktop applications.



Once the set of slides are ready, we can save the presentation as a storyboard. The Story Board should be stored at a shared location like a UNC path or a document library on a SharePoint portal etc. We can now link the storyboard back to a work item by first connecting to a specific team project on TFS 2012 and then selecting the work item using ID or a query.




Now we can assign this work item to any stakeholder or a Customer Representative. They only need MS PowerPoint installed to view the storyboard or to give comments about it in the work item fields.

We have now completed one loop of considering feedback from stakeholder or a representative of customer. Thus we have collected customer feedback on the application flow and user interface of the software, before we start building it. Let us now see the feedback capability after a feature is built and deployed on a staging server.

## CUSTOMER FEEDBACK REQUESTS

The Feedback Request feature requires that the SMTP Server be setup correctly on the TFS. This setting is used to send alerts and notifications raised by TFS. Without that, the option to create feedback request does not appear at all. This feature is used from TFS Project's Web Access. We will find the link to create feedback request on the home page of the Team Project's web access.

When a software feature is ready and deployed on a staging server, the development team can request the Customer Representative to provide feedback. This usually happens at the end of the iteration / sprint. Feedback provided by the customer will then be analyzed. If any changes are required, then a new user story or PBI will be created and considered for the subsequent iteration.

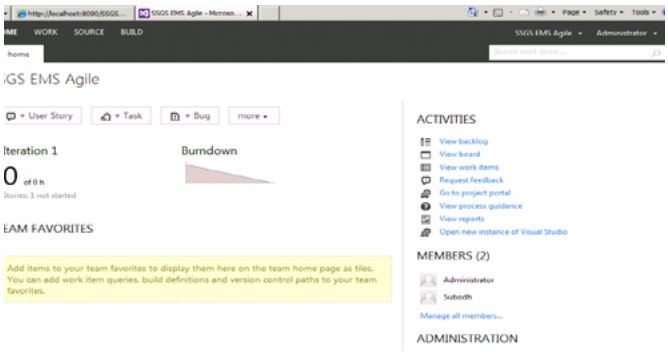
**Prerequisites for Customer Feedback integration with TFS 2012**

Customer can access the feedback request using a dedicated client that can be downloaded from Microsoft website http://www.microsoft.com/en-in/download/details.aspx?id=30660

Accessing feedback request does not need any additional license (TFS or VS 2012). The customer has to be provided the following privileges in TFS

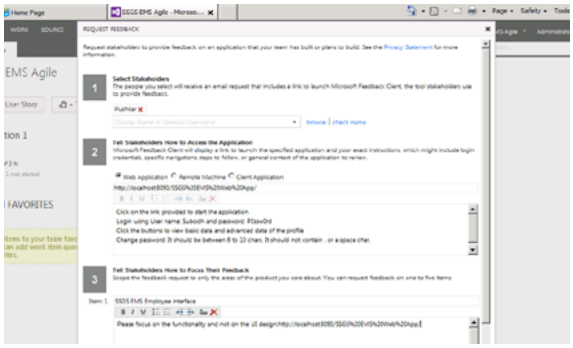
1. From the TFS Control Panel, the customer should at least be provided 'Limited Permissions'.
2. Under the Team Project, customer should have permission to View and Edit Work Items.
3. Customer should be granted permissions to Create Test Runs, View project-level information, and View Test Runs.

Now let us see how to create a feedback request. A link to do so appears on the home page of the team project in web access.

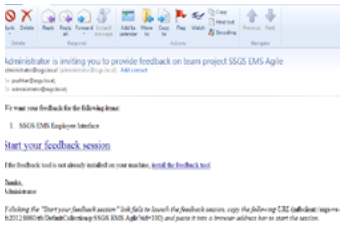



After selecting the link, a form appears where we can provide following data:

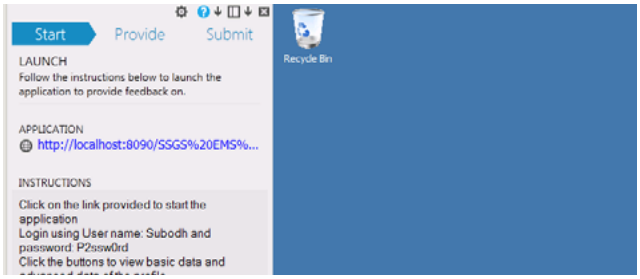
1. Name of the customer to whom the feedback request is to be sent
2. URL / Path from where the application is available
3. Application Launch instructions
4. Name of the application (free text)
5. Instructions to customer regarding focus area in the application and steps to execute.



Once we setup the required information and hit 'Send', email address of the customer is automatically picked up from the active directory and an email is sent to the customer. This email notifies the customer of the feedback request. This email also contains a link to launch the feedback client. Behind the scenes, a work item of the type 'Feedback Request' is automatically created.



When the customer clicks on that link in the email, the feedback client that is downloaded and installed on the customer's machine is launched. It connects to appropriate TFS and downloads the data from the work item. That data is shown in the left hand pane of the tool whereas the right pane is where the application may be run.




Once the application is launched, customer can go to the Next screen in the tool and provide feedback for any number of screens in the application, and the feedback can be in the form of Text, Screenshots, Video (Screen Capture) and audio. All of these are created as separate files that are linked to another work item of the type - Feedback Response. When customer submits the feedback, this work item is created. It is automatically assigned to the person who had initially requested the feedback. In this way, the feedback about software in the staging environment is received by the team. This can be reviewed and analyzed by the development team who can then make necessary changes.

## IN CONCLUSION

Visual Studio Team Foundation Server 2012 is an excellent facilitator for implementing various principles of Agile in general and SCRUM in particular. Early feedback is necessary from the point of view of acceptability of incremental changes, which satisfy the customer needs. Feedback received for the storyboards can help the team to build the right software that meets customer expectations. Feedback received over the built features allows the team to take early corrective actions and to get new requirements from the customer. These two additions in TFS 2012 helps round off the suite of tools and services required to implement Agile principles easily. ■

Subodh Sohoni, is a VS ALM MVP and a Microsoft Certified Trainer since 2004. Follow him on twitter @subodhsohoni and check out his articles on TFS and VS ALM at http://bit.ly/Ns9TNU

# Web Essentials

## Essential Add-on for Visual Studio

GREGOR SUTTIE WALKS US THROUGH THE FEATURES OF ONE OF THE MOST PRODUCTIVE PLUGINS FOR A WEB DEVELOPER WORKING IN VISUAL STUDIO 2102

Web Essentials is a free Visual Studio extension written by Mads Kristensen (Program Manager at the Microsoft Web Platform Team). It is designed to make your life easier and more productive as a web developer. It includes a large set of utilities and shortcuts that help when making changes to HTML, CSS, JavaScript and more. Today we will see more about what Web Essentials offers.

### WEB ESSENTIALS FEATURES

Web Essentials has a number of features which are being updated almost on a weekly basis. At the time of writing this article, the features to be found within Web essentials are as follows

- HTML Zen Coding within HTML
- CoffeeScript Support
- CSS Support
- JavaScript Support
- JSHint Support
- LESS Support
- TypeScript
- Other features

Let's discuss each of these features in turn, so we can get a good feel for what the extension gives us as a developer.


### HTML Zen Coding

HTML Zen Coding is an editor plug-in for high-speed frontend code editing - if you're a frontend web developer or if you are spending a large amount of time working with HTML, XML, or XSL, then it may be beneficial to have a look at this feature.

Zen Coding is developed by Sergey Chikuyonok and has been incorporated into a number of useful text editors over time which now includes Visual Studio. The Zen Coding project is available on Github.

You can learn more about Zen Coding on Wikipedia: and on Google: http://code.google.com/p/zen-coding/

When working with HTML a lot, it can be tedious to have to continually type html tags for everything in full and with HTML Zen Coding the Web Essentials gives us some nice little shortcuts, a few examples of these would be like so:

For an HTML 5 header you would type **html:5** and this would be added:

```
<!DOCTYPE HTML>
<html lang=”en-EN”>
<head>
<title></title>
<meta charset=”UTF-8”>
</head>
<body>
</body>
</html>
```

Similarly if you wanted to add a new password input field:-

**input:p** then tab or **input:password**

There are shortcuts for almost every html tag available and the cheat sheet below will provide much more information: http://code.google.com/p/zen-coding/wiki/CheatSheets

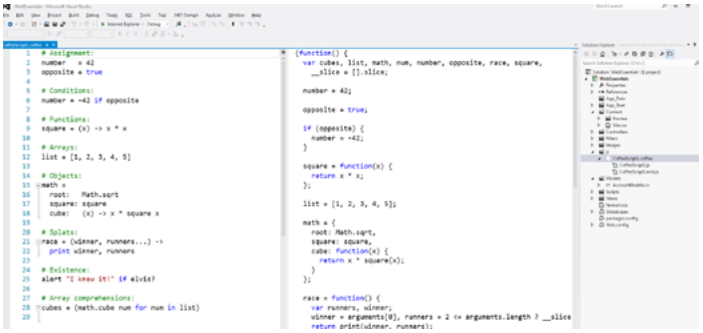
Editor's Note: While we were getting this article prepped, the Zen Coding project transformed into Emmet. You can follow the project now at https://github.com/emmetio/emmet.

### CoffeeScript Support

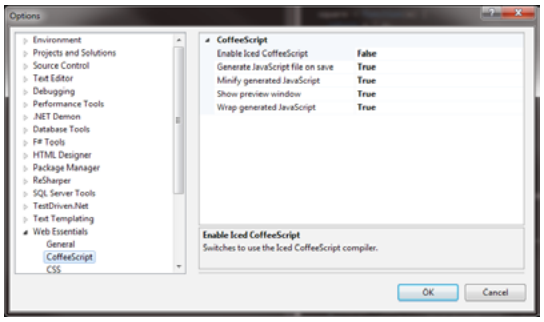
CoffeeScript is a language it its own right which compiles into JavaScript and the Web Essentials extension has very sweet support for this. When you are editing your CoffeeScript file within Visual Studio, the window you're working with is split into two, so that you can see if the JavaScript produced, looks good.

You can learn more about CoffeeScript: http://coffeescript.org/

The screenshot below shows how Web Essentials takes a CoffeeScript file and as you type into the left hand pane, the add-on will convert your code to JavaScript and display what the code will look like once you save the file.

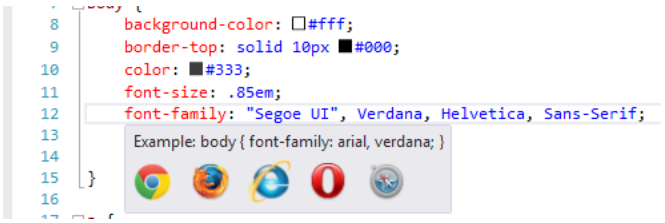


Support for CoffeeScript is included with Web Essentials and the following screen shot shows us the options available

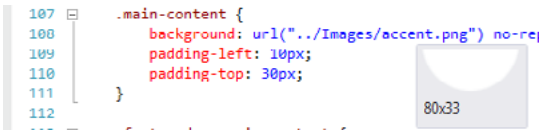



## CSS Support

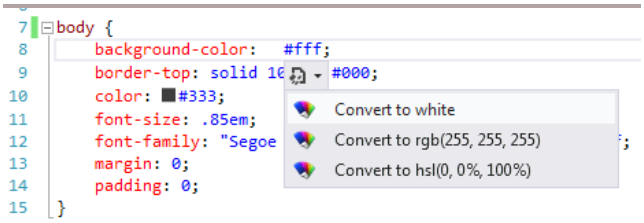
Web Essentials gives us some really useful features for developers when you're working with CSS, if you open a css file to add some new styles or even check existing css styles, and you hover the mouse over the style, Web Essentials will display something like the following



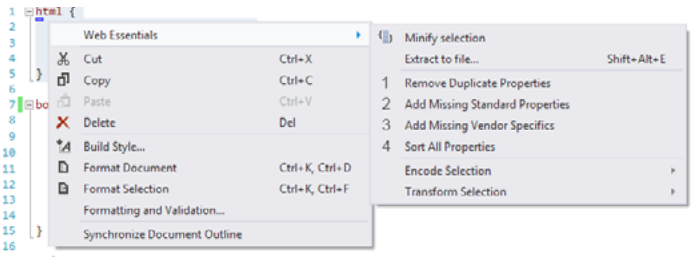

Here the add-on is telling us that the font-family: arial, verdana; works in the list of browsers represented by the icons shown. Hovering over an image will also display the image as follows:



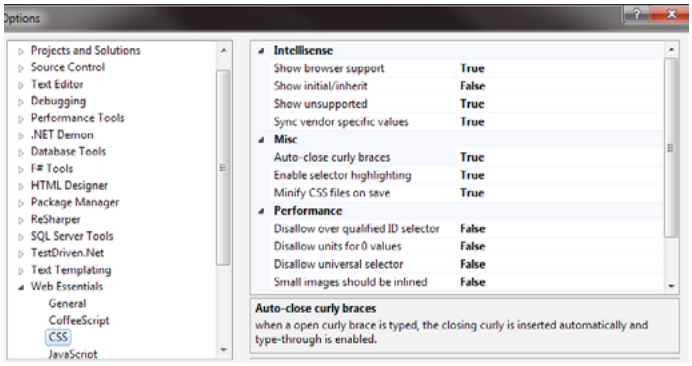

This time the add-on is informing us that we convert the background color:




Here I have right clicked on a css element and we can see the options Web Essentials gives us, we can minify the selection, as well as remove/add standard properties and also sort them.




Support for CSS is included with Web Essentials and the following screen shot shows us what is available:

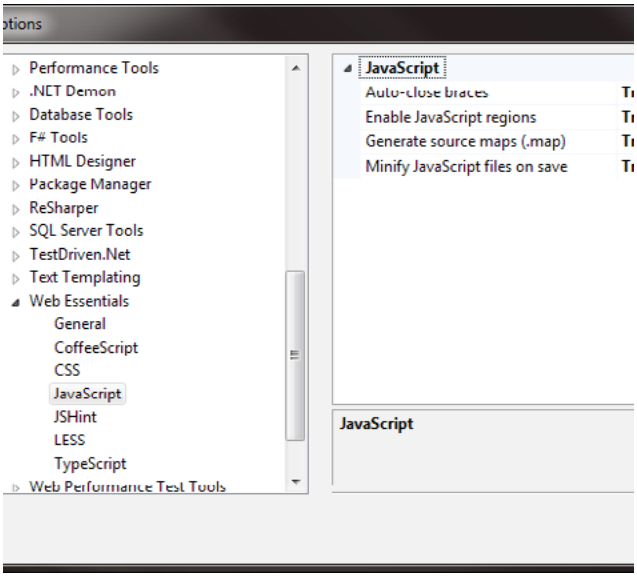



A very handy little piece of information is that the following website http://realworldvalidator.com/ is used by Web Essentials to keep track of css standards and what styles will work on each browser. The follow screen grab is from the website:

Here we can see a list of what works on each browser. This can be very handy when we are working on a project and we know we have a checklist to work against when testing against specific browsers, and even specific browser versions.

## JavaScript Support

There are a set of JavaScript enhancements included with Web Essentials too. Following screen-shot shows us what is available:




Simple add-ons but very handy when working with JavaScript files for any length of time, auto-close braces is especially useful when your editing larger chunks of JavaScript.

Turn on minify JavaScript on save when about to release and leave it off for when you are debugging - simple but effective features.

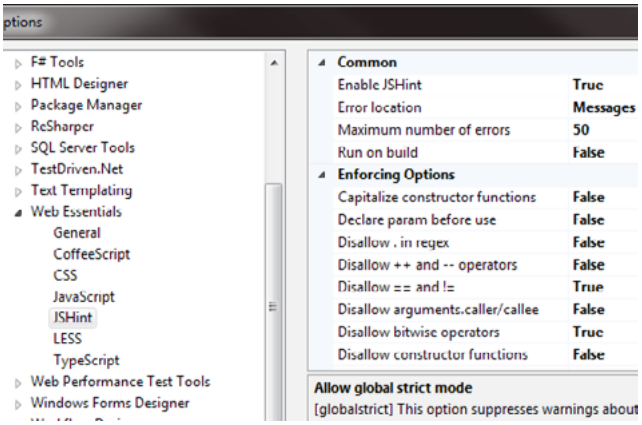
## JSHint Support

***"JSHint is a tool to detect errors and potential problems in JavaScript and can be used to enforce coding conventions"*** – you can tailor what JSHint will check for to meet your own needs or even your teams coding standards for JavaScript code.

"***The goal of JSHint is to help JavaScript developers write complex programs without worrying about typos and gotchas.***" – taken from JSHint.com

You can learn more about JSHint: http://www.jshint.com/

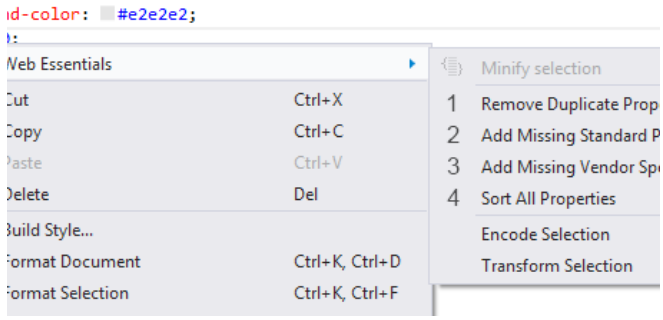
JSHint integration using Web Essentials gives us a lot of options to tailor, the following is a screen shot of just some of the options available for us to change if we see fit:



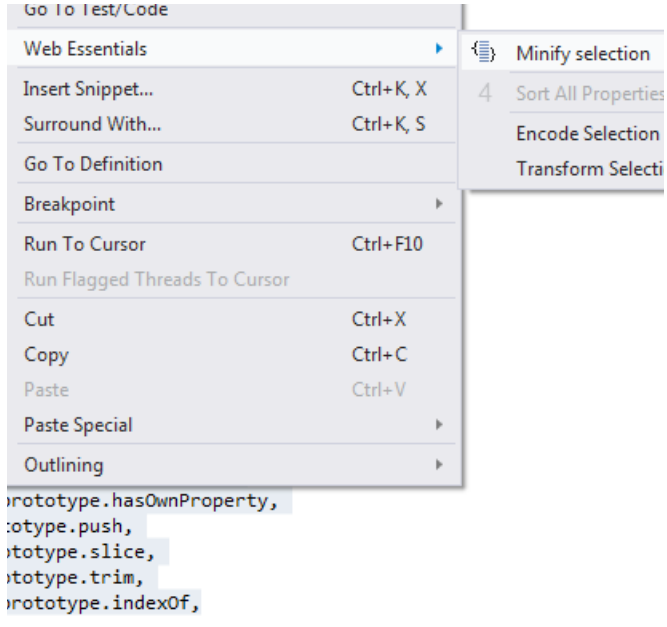

## Other Features

Web Essentials comes with a whole host of other goodies and these include when you're editing particular file types, let's take a look at what's available

When working with a .ccs file, you can right click on a style and get options relevant to the context as seen below:




When working with a .js file, you can right click on a block of JavaScript and get options relevant to the context as seen below:

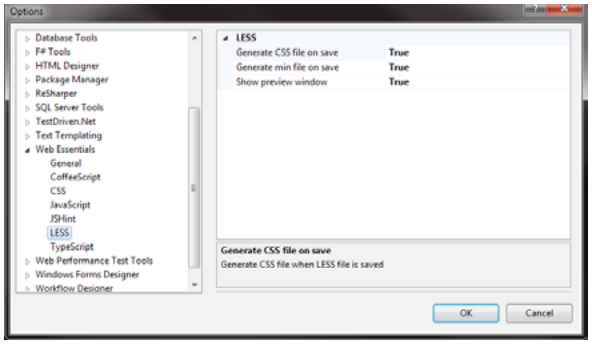


### LESS Support

LESS is a dynamic stylesheet language which allows us to extend CSS with dynamic behavior using variables, operations, functions and more. Aim of LESS is to manage your CSS better, however client side involves a JavaScript interpreter. Web Essentials on the other hand helps compile LESS into CSS at design time itself.

You can learn more about LESS: http://lesscss.org/

Support for LESS is included with Web Essentials and the following screen shot shows us what is available:

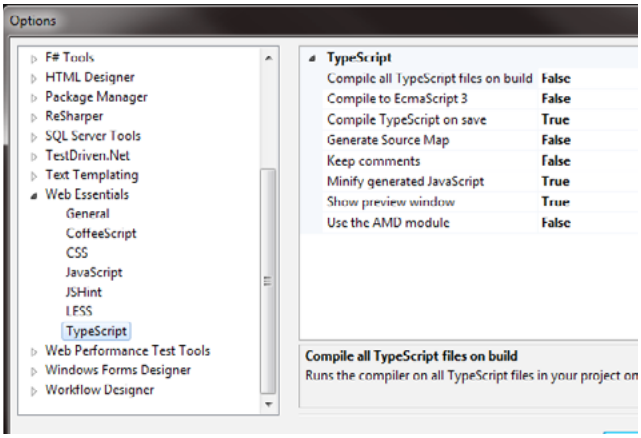



### TypeScript Support

***"TypeScript is a language for application-scale JavaScript development. TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. Any browser. Any Host. Any OS. Open Source"-*** source http://www.typescriptlang.org/

TypeScript is a project currently led by Microsoft Fellow Anders Hejlsberg. It's again a language that compiles down to JavaScript. However typescript itself is EcmaScript draft v6 compliant. Learn more about TypeScript at http://www.typescriptlang.org/

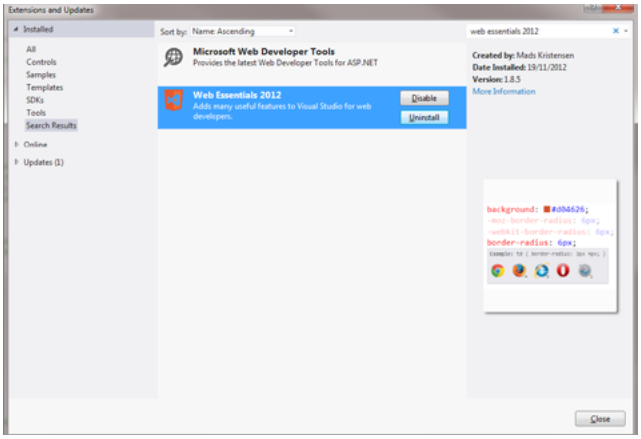
Support for TypeScript is included with Web Essentials and the following screen shot shows us what is available:




TypeScript support is fairly new and is a very good example of why Web Essentials is a great extension to Visual Studio; it makes it easier to add in new features rather than having it baked into Visual Studio allowing the creator to add more functionality on a far more regular basis.

## HOW DO I GET WEB ESSENTIALS?

To install the Web Essentials add-on for Visual Studio 2012 you can grab it either from here: http://visualstudiogallery.msdn.microsoft.com/07d54d12-7133-4e15-becb-6f451ea3bea6
Or you can grab it from inside Visual Studio 2012 by browsing to the Tools Menu and then selecting Extensions and Updates, then search for Web essentials, an example of the screen you will see within Visual Studio is as follows:

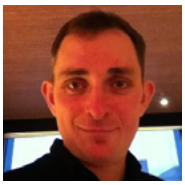



You can also learn more about it here: http://visualstudiogallery.msdn.microsoft.com/07d54d12-7133-4e15-becb-6f451ea3bea6 where you can suggest new features and report bugs.

Mads Kristensen has been updating this on a regular basis so keep an eye out for updates within Visual Studio for this extension.

## SUMMARY

Web Essentials is a fantastic add-on for Visual Studio and if you're doing any kind of front end web development work then the add-on is as the name suggests essential. Its free, regularly updated and I highly recommend you have a look. You can follow Mads on twitter here http://twitter.com/mkristensen. ■



Gregor Suttie is a developer who has been working on mostly Microsoft technologies for the past 14 years, he is 35 and from near Glasgow, Scotland. You can Follow him on twitter @gsuttie and read his articles at bit.ly/z8oUjM

# Designing a Lossless Information System

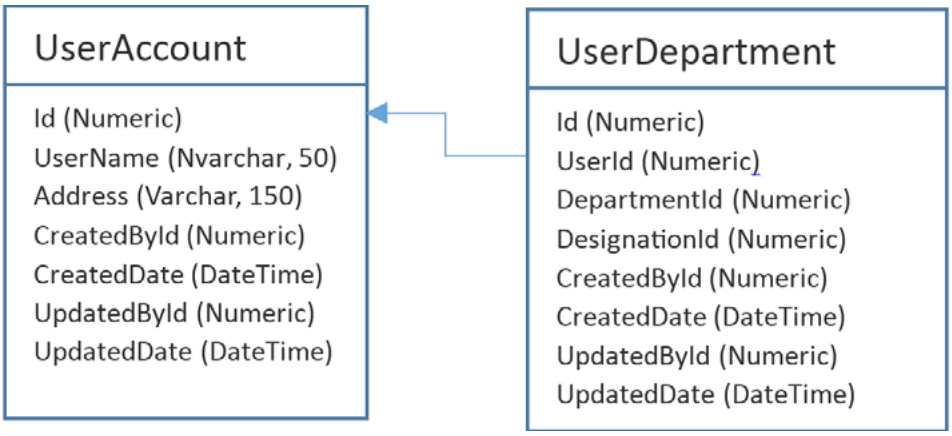
**Sumit introduces a database design pattern that ensures data once inserted into the database is never lost, a must have for financial systems**

Almost a year back Clemens Vasters (Architect at Microsoft) Tweeted saying both Updates and Deletes (in databases) result in Destruction of data. This struck me as a very interesting take on maintenance of Historical data. It turns out it's not an Architectural fantasy but a hard bound requirement in the Finance world where 'destruction' of any type of data is deemed a criminal offense.

Ever since that Tweet from Clemens, I have been meaning to explore this pattern. Finally, when suggestions for an Architecture focused article in the magazine popped up, I decided to do a deep dive in what I refer to as - a Lossless Information System design.

A Lossless Information System (LLIS) is one that ensures that every data operation in it, is tracked and no information is ever deleted (archived but not deleted). Projects in Finance domain are the biggest targets for Lossless Information Systems because of stringent Audit trail requirements. This type of data is often referred to as immutable data.

Today we will see how we can build such a system using a traditional RDBMS.




## A LITTLE HISTORY

A LLIS is not a new concept and has be academically researched extensively in the 1980s and 90s. One of the notable works on LLIS is an IEEE paper by Bhargava, G and Gadia S.K. Their paper was published in IEEE journal – IEEE Transactions on Knowledge and Data Engineering in Feb 1993 and this article leans heavily on the paper for theoretical relevance.

## THE SAMPLE SCHEMA

For simplicity, we will use the scenario where we need to save User Account information in a way that every change is tracked, leaving us an easy way to trace history trail. Our Sample schema will save User Account + UserDepartment association information.

As we see above, the two tables store information about a user and their department. [UserDepartment.UserID] is the foreign key (FK) to the UserAccount table

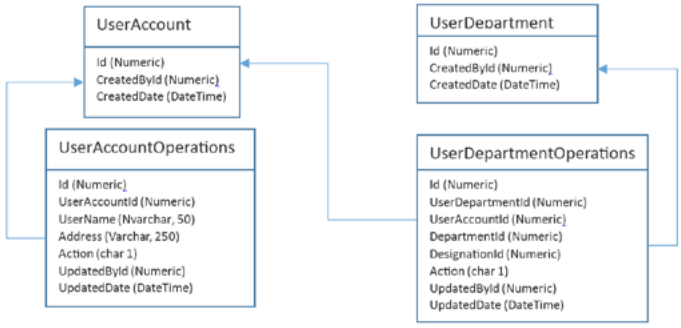
### Traditional Usage and limitations

Traditional Usage of the above schema is to simply timestamp on creation and modify the UpdatedBy timestamp at the time of updates. It provides a creation history and the last update history. It depends on backend archival and backup systems to retrieve old data. This suffices where audit requirements do not mandate exhaustive history. However, Finance domain audit

requirements often mandate that data once entered in a system, should never be destroyed. This implies every bit of data be traceable. This gives us the initial impression that data cannot be deleted or in other words the DELETE operation is restricted. However, if we look deeper, we will realize that even updating a value from False to True is in fact 'destruction' of data. The value of False is now 'Replaced' and hence no longer retrievable and thus considered 'destroyed'. This extends our restrictions to DELETE and UPDATE operations.

In a world of transactional RDBMS when CREATE, READ, UPDATE and DELETE is the norm, taking out UPDATE and DELETE kind of throws a major spanner in the works, at least on the outset. Let's see what we can do to try and conform to this 'requirement'.

## MODIFYING SCHEMA FOR HISTORY TRACKING




As we see above, we have split each table into an Id table and an Operations table. The ID table has what is referred to as non-volatile data (because a User.Id value is non-immutable throughout the lifetime of the entity). The volatile data, data that is updated in the lifetime of the entity, is moved to the Operations table. Also notice the Action column that has been added to the volatile data table. This can have three values C(reate), U(pdate) and D(elete), depending on the action. Id is the auto-increment PK.

### Database Operations

Given the above changes, let's see how we can do the CRUD operations or their equivalents.

#### CREATE Operation

This involves Inserting a row of data in the Id table. This gives us the Primary Key to be used in the Operations table. Once we have the Primary Key, we insert the volatile data in the Operations table with the action flag value of 'C' (for create).

#### UPDATE Operation

To update the data entered above, we do another INSERT operation. However we need to touch (the volatile) data that's in the operations table only. So an UPDATE operation actually does an INSERT with the Action flag set to 'U'. All fields that have been updated in that operation are INSERTed in the operations table.

#### DELETE Operation

To delete data, we essentially mark an entry as deleted by inserting a row in the operations table with the Action flag 'D'.

#### READ Operation

Now that we have seen all the update operations, we have one challenge to overcome. That is SELECT. Since we have multiple possible rows of volatile data (in the operations table) for each non-volatile data item, the SELECT operation needs to be able to select the Active or the most recent 'row' of data. Max value of the Id column in the operations table, for a given Foreign Key (UserAccountId or UserDepartmentId) will always have the latest value. So in our filter, we check for the Max value of the Id of the Operations table for each row in the Id table. Next we check if the latest field's Action is 'D' and exclude all rows marked as such. This returns us the last created/updated data that is not deleted. For easy access to this set of data, we create a View on top of the Id and Operations tables with the above filtering constraints.

## IMPLEMENTING THE LLIS

### Verifying data at SQL Server

Before we get into MVC and EntityFramework, let's validate our design by entering some data in our tables. Let's say we want to Create a new User 'Sumit' at location 'Pune', we would fire the following two queries

```
INSERT INTO UserAccounts (CreatedById, CreatedDate)
Values(1, GETDATE());

INSERT INTO UserAccountOperations
           ([UserAccountId]
           ,[UserName]
           ,[Address]
           ,[Action]
           ,[UpdatedById]
           ,[UpdatedDate])
     VALUES
           (@@IDENTITY
           ,'Sumit'
           ,'Pune'
           ,'C'
           ,1
           ,GETDATE())
```

The UserAcccountOperations table is using the Identity key from the UserAccounts table and updating itself with the required data and status flag as 'C'

Once this data is entered and we execute the following query:

```
SELECT
UserAccounts.Id, UserAccounts.
CreatedById, UserAccounts.
CreatedDate, UserAccountOperations.
Id
AS
OperationsId, UserAccountOperations.UserName,
UserAccountOperations.Address, UserAccountOperations.
Action, UserAccountOperations.UpdatedById,
UserAccountOperations.UpdatedDate
FROM
UserAccountOperations INNER JOIN UserAccounts ON
UserAccountOperations.UserAccountId = UserAccounts.Id
WHERE
(UserAccountOperations.Action <> 'D') AND
(UserAccountOperations.Id =
 (SELECT MAX(Id) AS MaxId
  FROM UserAccountOperations AS UAO
  WHERE (UserAccountId = UserAccountOperations.
UserAccountId)
 )
)
```

This query returns the following data

| | Id | CreatedById | CreatedDate | OperationsId | UserName | Address | Action | UpdatedById | UpdatedDate |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2012-12-27 05:09:12.020 | 1 | Sumit | Pune | C | 1 | 2012-12-27 05:09:12.030 |

As we can see, the required data is coming up correctly. Next, we do an Update, where we change the Address from Pune to Mumbai. As discussed above, we use an INSERT into the Operations table with the Action flag 'U' using the following query.

```
INSERT INTO UserAccountOperations
            ([UserAccountId]
            ,[UserName]
            ,[Address]
            ,[Action]
            ,[UpdatedById]
            ,[UpdatedDate])
      VALUES
            (1,
            ,'Sumit'
            ,'Mumbai'
            ,'U'
            ,1
            ,GETDATE())
```

Now we have two rows in the Operations table for the same user, but if we use the query for our View, we will get only the last UPDATED row.

| | Id | CreatedById | CreatedDate | OperationsId | UserName | Address | Action | UpdatedById | UpdatedDate |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2012-12-27 05:09:12.020 | 3 | Sumit | Mumbai | U | 1 | 2012-12-27 10:44:58.783 |

To delete this row of data, we again do an INSERT operation into UserAccountOperations table with the ActionFlag set to 'D'.

```
INSERT INTO UserAccountOperations
            ([UserAccountId]
            ,[UserName]
            ,[Address]
            ,[Action]
            ,[UpdatedById]
            ,[UpdatedDate])
      VALUES
            (@@IDENTITY
            ,'Sumit'
            ,'Mumbai'
            ,'D'
            ,1
            ,GETDATE())
```

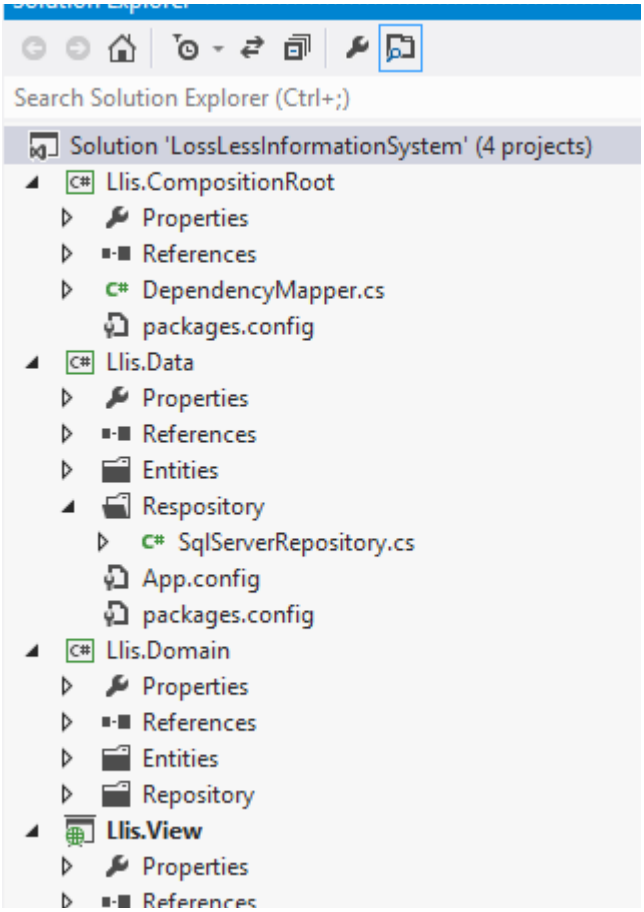
Now when we use the 'View' query, we end up with no rows of data even though we actually have three historic rows of data. The following query shows us the History

```
SELECT * FROM UserAccountOperations WHERE UserAccountId
= 1
```

| | Id | UserAccountId | UserName | Address | Action | U |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | Sumit | Pune | C | 1 |
| 2 | 3 | 1 | Sumit | Mumbai | U | 1 |
| 3 | 4 | 1 | Sumit | Mumbai | D | 1 |

## CREATING AN ASP.NET APPLICATION FOR A PROOF OF CONCEPT

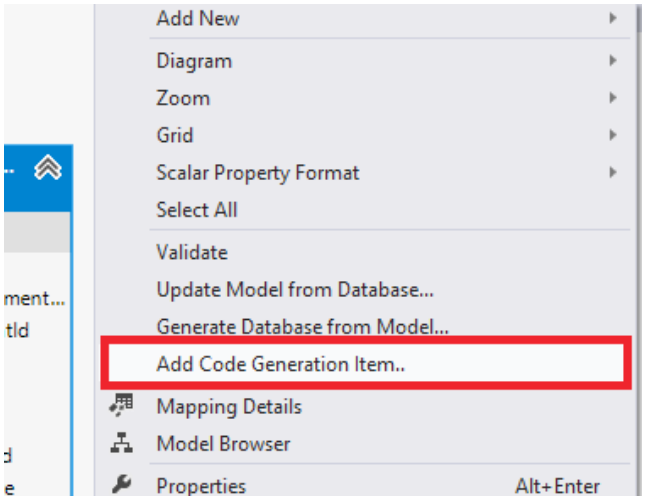
We create an ASP.NET WebAPI Solution and add three Class library projects to it.



Along with the View project, we have the CompositionRoot for mapping dependencies, Data for interacting with Database using EntityFramework, Domain layer for defining Business Entities and constraints. In the View layer, we make use of Ninject as our Dependency Injection framework.

## Generating the Data Layer using EF

Since we designed the DB Schema first, we'll use the Entity Framework EDMX Designer to reverse engineer our data layer. Once we have the DB Diagram, we right click on it and add the Code Generation T4 file EF5.x Db Context Generator. This T4 template ensures the entities generated via the EMDX are EF Code First compliant.




## Setting up Repository

In the Domain class, we'll setup the Repository Interface ISqlServerRepository

```
public interface IUserDataRepository
{
 User CreateUser(User newUser);
 User UpdateUser(User existingUser);
 User SelectUser(int userId);
 IEnumerable<User> SelectAllUsers();
 IEnumerable<User> SelectAllUsersHistory();
 IEnumerable<User> SelectUserHistory(int userId);
 bool DeleteUser(int userId);

 UserDepartment CreateUserDepartment(UserDepartment
newUser);
 UserDepartment UpdateUserDepartment(UserDepartment
existingUser);
 UserDepartment SelectUserDepartment(int userId);
 IEnumerable<UserDepartment> SelectAllUserDepartments();
 bool DeleteUserDepartment(int userId);
}
```

Here User and UserDepartment are domain objects that don't map to the database schema, instead they encapsulate data that the User and UserDeparment entities should contain from a business perspective.

In the Llis.Data library that contains the EF Code first data objects, we implement the IUserDataRepository in SqlServerRepository class. This is where we deviate from traditional implementations. Instead of using the DBContext to call the respective operations, we change the implementation such that the Updates and Deletes are replaced by Insert statements and the Read/Select operation uses the View Poco to retrieve required data.

## Setting up Ninject

To effectively use the Repository pattern, we will go with a DI framework. Ninject is pretty easy to use and get started with. In the Llis.View project add the Ninject package via Nuget. The Console command is

**Install-package Ninject**

Next we add the MVC extensions, these are a part of the Ninject.MVC3 package. Don't be alarmed they work with MVC4 just fine.

**Install-package Ninject.MVC3**

Now that these two dependencies are in place, open the App_Start\NinjectWebCommon.cs file that's added by the Ninject.MVC3 package. This class is responsible for hooking Ninject on to MVC as the DI Framework.

Here update the empty RegisterServices method as follows

```
private static void RegisterServices(IKernel kernel)
{
 kernel.Load(Assembly.GetExecutingAssembly(),
  Assembly.Load("Llis.Domain"),
  Assembly.Load("Llis.Data"),
  Assembly.Load("Llis.CompositionRoot"));
}
```

We are providing Ninject will all possible dependencies in that might have the Concrete implementations of interfaces that we may need.

Next in the Llis.CompositionRoot project we add the NInject package first and then add a class called DependencyMapper as follows

```
public class DependencyMapper : NinjectModule
{
 public override void Load()
 {
  this.Bind<IUserDataRepository>().
To<SqlServerRepository>();
```

```
 }
}
```

The CompositionRoot library is traditionally where an IoC container composes objects. In our case, the Interface and Implementation are in different libraries (justifiably so) as we don't want the View layer to carry any direct dependency of the Data layer. We use the CompositionRoot to bridge this problem.

Please note, all Interfaces whose concrete instances are required need to be registered here.

## Setting up an Index and History View

Now that we have setup the backend, let's create a couple of Views, one for the Index and one for the History of User data. The data is returned to us via the

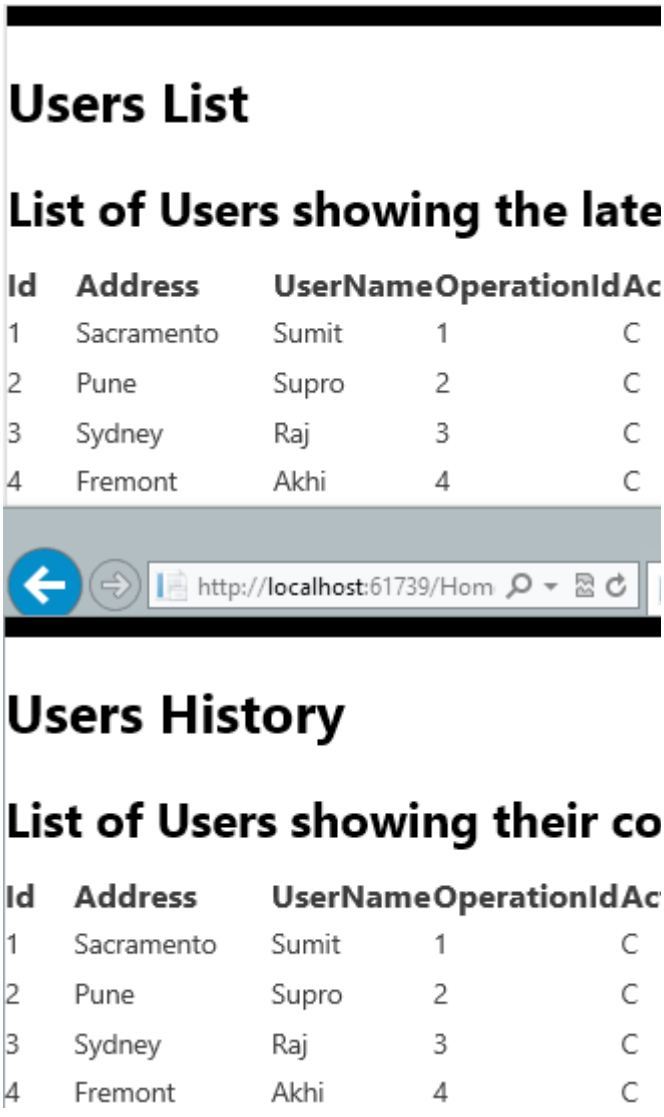
```
IEnumerable<User> SelectAllUsers();
IEnumerable<User> SelectAllUsersHistory();
```

implementations in the repository. Their implementations are as follows:

```
public IEnumerable<Domain.Entities.User>
SelectAllUsersHistory()
{
 List<Domain.Entities.User> entities = new List<Domain.
Entities.User>();
using (llisdb _context = new llisdb())
{
 foreach (var item in _context.UserAccounts.Include("Use
rAccountOperations"))
 {
  foreach (var operItem in item.UserAccountOperations)
  {
   entities.Add(new Domain.Entities.User
   {
    Id = item.Id,
    Address = operItem.Address,
    OperationId = operItem.Id,
    UserName = operItem.UserName,
    Action = operItem.Action,
    CreateDate = item.CreatedDate,
    UpdatedDate = operItem.UpdatedDate
    });
   }
  }
 }
 return entities.OrderBy(k => k.OperationId);
}
public IEnumerable<Domain.Entities.User> SelectAllUsers()
{
 List<Domain.Entities.User> entities = new List<Domain.
Entities.User>();
 using (llisdb _context = new llisdb())
 {
  var items = _context.UserAccountsViews.
ToList<UserAccountsView>();
  foreach (var operItem in items)
  {
   entities.Add(new Domain.Entities.User
   {
    Id = operItem.Id,
    Address = operItem.Address,
    OperationId = operItem.OperationsId,
    UserName = operItem.UserName,
    Action = operItem.Action,
    CreateDate = operItem.CreatedDate,
    UpdatedDate = operItem.UpdatedDate
   });
  }
 }
 return entities;
}
```

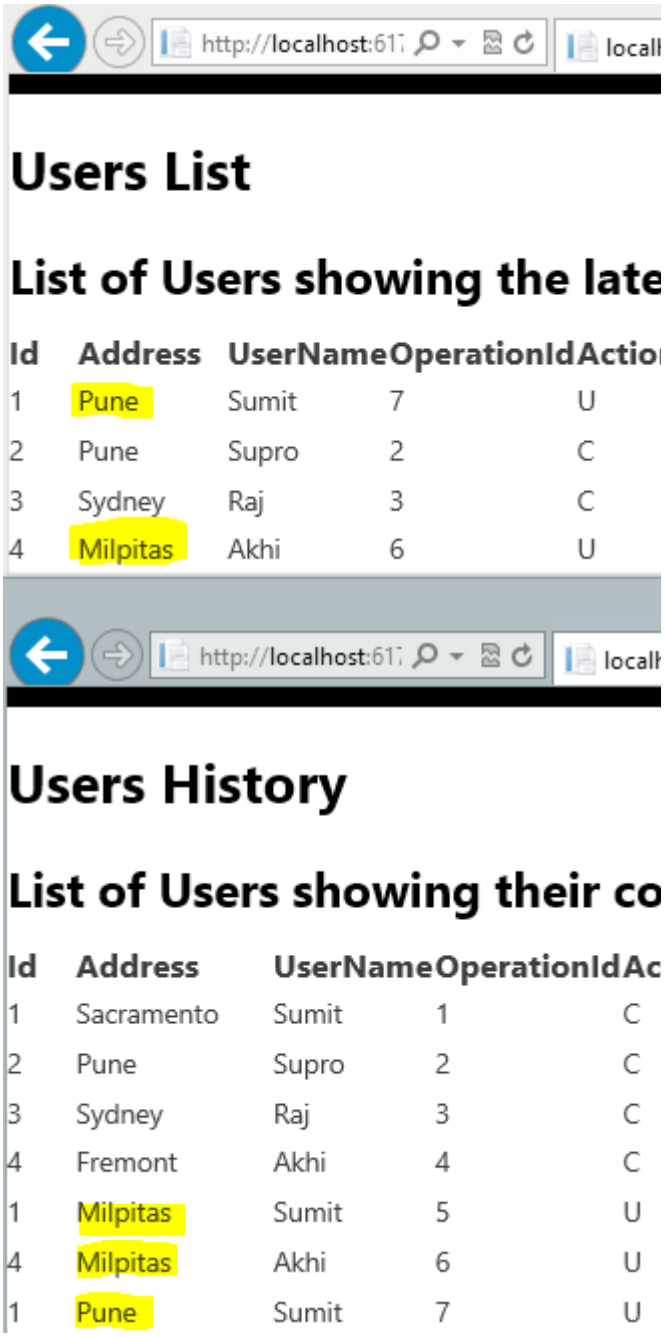
As we can see, SelectAllUsersHistory actually fetches data using the User (and UserOperations) tables. This returns us the complete history of the data. SelectAllUsers on the other hand, uses the UserAccountView that returns only the latest data.

From the given script (UserData.sql) if we run the INSERT queries only, we get data as follows:



As we can see in the image, there are 4 Users and they have four corresponding entries in the history table. All their actions are 'C' (for Create).

Next if we run the Update portion of the included script and refresh the above two pages, we'll see data as shown below



Users List

List of Users showing the late

| Id | Address | UserName | OperationId | Actio |
|---|---|---|---|---|
| 1 | Pune | Sumit | 7 | U |
| 2 | Pune | Supro | 2 | C |
| 3 | Sydney | Raj | 3 | C |
| 4 | Milpitas | Akhi | 6 | U |

Users History

List of Users showing their co

| Id | Address | UserName | OperationId | Ac |
|---|---|---|---|---|
| 1 | Sacramento | Sumit | 1 | C |
| 2 | Pune | Supro | 2 | C |
| 3 | Sydney | Raj | 3 | C |
| 4 | Fremont | Akhi | 4 | C |
| 1 | Milpitas | Sumit | 5 | U |
| 4 | Milpitas | Akhi | 6 | U |
| 1 | Pune | Sumit | 7 | U |

As highlighted in the Users List, the User 'Sumit' seems to have moved to 'Pune' and the User 'Akhi' has moved to 'Milpitas'.

However if we check the history table, User 'Sumit' in fact made a pit stop at 'Milpitas' before heading out to 'Pune'. We can also see the time at which the changes were made and in the database we have the UserId to show who made the change.

As a result, we now have an easy to maintain Audit trail of our User Data.

## CONCLUSION

Trying to maintain a lossless system has many approaches like copying data over to archive before updating or using built in database transaction logs and so on. When business requirement impose strict restrictions on what DML operations can be allowed, the design we just explored can be used to build a simple Loss Less Information System.

There are two major considerations, first being volume of transactions and second being performance. There will be a trade-off, however by baking in reasonable performance expectations with stringent archival policy can ensure Performance never goes below given SLA and system remains responsive for the longest period. ■

Source code for this sample can be downloaded from Github at http://bit.ly/dncm-04-llis

Sumit is a .NET consultant and has been working on Microsoft Technologies for the past 12 years. He edits, he codes and he manages content when at work. C# is his first love, but he is often seen flirting with Java and Objective C. You can Follow him on twitter @sumitkm and read his articles at bit.ly/KZ8Zxb