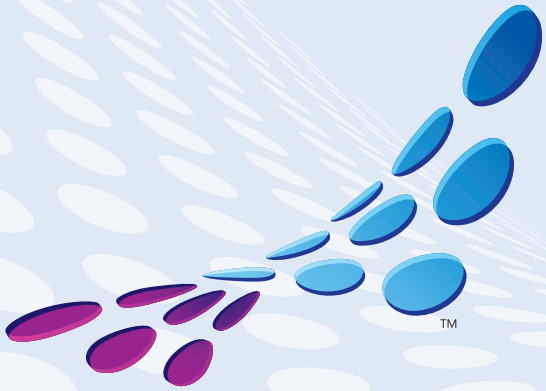
THE MICROSOFT JOURNAL FOR DEVELOPERS

AUGUST 2010 VOL 25 NO 8



msdn magazine

DEALING WITH DATA

COLUMNS

Microsoft

msdn® magazine

AUGUST 2010 VOLUME 25 NUMBER 8

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in U.S. funds: U.S. $35; Canada $45; International $60. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: MSDN Magazine, PO. Box 3167, Carol Stream, IL 60132, e-mail MSDNmag@1105service.com or call 847-763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, PO. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or IMS/NJ. Attn: Returns, 310 Paterson Plank Road, Carlstadt, NJ 07072.

Corporate Address: 1105 Media, Inc., 9201 Oakdale Ave., Ste 101, Chatsworth, CA 91311, www.1105media.com

Media Kits: Direct your Media Kit requests to Matt Morollo, VP Publishing, 508-532-1418 (phone), 508-875-6622 (fax), mmorollo@1105media.com

Reprints: For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International, Phone: 212-221-9595, E-mail: 1105reprints@parsintl.com, www.magreprints.com/QuickQuote.asp

List Rental: This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Merit Direct. Phone: 914-368-1000; E-mail: 1105media@meritdirect.com; Web: www.meritdirect.com/1105

All customer service inquiries should be sent to MSDNmag@1105service.com or call 847-763-9560.

Microsoft

BPA WORLDWIDE

Printed in the USA

Your best source for software development tools!

programmer's paradise

NEW RELEASE!

LEADTOOLS Recognition SDK

by LEAD Technologies

Develop desktop and server document imaging and ECM applications that require high-speed multi-threaded forms recognition and processing, OCR, ICR, OMR, and barcode technology.

- Supports text, OMR, image, and barcode fields
- Recognize machine print and constrained handwritten text
- Auto-registration and clean-up to improve recognition results
- Includes comprehensive confidence reports to assess performance

Paradise # L05 26301A01

$3,214.99

programmers.com/LEAD

Multi-Edit^x

NEW RELEASE!

by Multi Edit Software

Multi-Edit^x is "The Solution" for your editing needs with support for over 50 languages. Edit plain text, ANY Unicode, hex, XML, HTML, PHP, Java, Javascript, Perl and more! No more file size limitations, unlimited line length, any file, any size Multi-Edit^x is "The Solution"!

Pre-Order Your Copy and Save!

1-49 Users
Paradise # A30Z10101A01

$223.20

programmers.com/multiedit

VMware vSphere

Put time back into your day.

Your business depends on how you spend your time. You need to manage IT costs without losing time or performance. With proven cost-effective virtualization solutions from VMware, you can:

- Increase the productivity of your existing staff three times over
- Control downtime—whether planned or not
- Save more than 50% on the cost of managing, powering and cooling servers

Make your time (and money) count for more with virtualization from VMware.

VMware Advanced Acceleration Kit for 6 processors
Paradise # V55 78101A01

$9,234.99

programmers.com/vSphere

Certified for Windows 7/2008R2

"Pragma SSH for Windows" Best SSH/SFTP/SCP Servers and Clients for Windows

by Pragma Systems

Get all in one easy to use high performance package. FIPS Certified and Certified for Windows.

- Certified for Windows Server 2008R2
- Compatible with Windows 7
- High-performance servers with centralized management
- Active Directory & GSSAPI authentication
- Supports over 1000 sessions
- Hyper-V and PowerShell support
- Runs in Windows 2008R2/2008/2003/7/Vista/XP/2000

Paradise # P35 04201A01

$550.99

programmers.com/pragma

InstallShield Professional for Windows

by Flexera Software

If your software targets Windows®, InstallShield® is your solution. It makes it easy to author high-quality reliable Windows Installer (MSI) and InstallScript installations and App-V™ virtual packages for Windows platforms, including Windows 7. InstallShield, the industry standard for MSI installations, also supports the latest Microsoft technologies including Visual Studio 2010, .NET Framework 4.0, IIS7.0, SQL Server 2008 SP1, and Windows Server 2008 R2 and Windows Installer 5, keeping your customers happy and your support costs down.

FLEXERA SOFTWARE™ InstallShield

Upgrade from Active IS Pro + IS Pro Silver Mtn
Paradise # I21 02301S01

$1,399.00

programmers.com/flexera

VMware ThinApp Windows 7 Migration Promotion

by VMware

Migration to the new Windows 7 OS is forcing companies to rethink their desktop delivery. VMware ThinApp is the easiest and most cost effective way to prepare for your Win 7 journey. By virtualizing your applications first with ThinApp, you will mitigate risk, dramatically speed up the migration process—and save money when you migrate to Windows 7!

SAVE 75%!

LIMITED TIME OFFER!

Client Licenses Minimum 500 with Basic Level Support
Paradise # V55 MIGRATION

$16.99

programmers.com/vmware

NEW VERSION 6!

ActiveReports 6

by GrapeCity

Integrate Business Intelligence/Reporting/Data Analysis into your .NET applications using the NEW ActiveReports 6.

- Fast and Flexible reporting engine
- Data Visualization and Layout Controls such as Chart, Barcode and Table Cross Section Controls
- Wide range of Export and Preview formats including Windows Forms Viewer, Web Viewer, Adobe Flash and PDF
- Royalty-Free Licensing for Web and Windows applications

Professional Ed.
Paradise # D03 04301A01

$1,310.99

programmers.com/grapecity

Adobe Creative Suite 5 fro Windows

by Adobe

Discover breakthrough interactive design tools that enable you to create, deliver, and optimize beautiful, high-impact digital experiences across media and devices. Create once and deliver that same experience virtually everywhere, thanks to the highly anticipated releases of the Adobe® Flash® Player 10.1 and Adobe AIR® 2 runtimes. Maximize the impact of what you've created through integration of signature Omniture® technologies.

NEW VERSION 5!

Paradise # A14 06201A02

$2,555.99

programmers.com/adobe

TX Text Control 15.1

Word Processing Components

NEW RELEASE!

TX Text Control is royalty-free, robust and powerful word processing software in reusable component form.

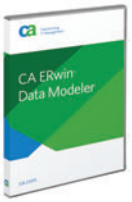
- .NET WinForms control for VB.NET and C#
- ActiveX for VB6, Delphi, VBScript/HTML, ASP
- File formats DOCX, DOC, RTF, HTML, XML, TXT
- PDF and PDF/A export, PDF text import
- Tables, headers & footers, text frames, bullets, structured numbered lists, multiple undo/redo, sections, merge fields, columns
- Ready-to-use toolbars and dialog boxes

Professional Edition
Paradise # T79 02101A02

$1,220.99

Download a demo today.

programmers.com/theimagingsource



CA ERwin® Data Modeler r7.3 – Product Plus 1 Year Enterprise Maintenance

by CA

CA ERwin Data Modeler is a data modeling solution that enables you to create and maintain databases, data warehouses and enterprise data resource models. These models help you visualize data structures so that you can effectively organize, manage and moderate data complexities, database technologies and the deployment environment.

Paradise # P26 04201E01

$3,931.99

programmers.com/ca

Intel Parallel Studio

by Intel

Intel Parallel Studio is a comprehensive Windows parallelism toolset designed for Microsoft Visual Studio C/C++ developers. Parallel Studio is interoperable with the widely used Microsoft Visual Studio, supports higher-level parallelism abstractions to simplify and speed development such as Intel Threading Building Blocks and Open MP, is fully supported, and provides an immediate opportunity to realize the benefits of multicore platforms. Tools are designed so novices can learn as they go, and professional developers can more easily bring parallelism to existing and new projects. Create optimized, innovative parallel applications and compete in a multicore industry.

Single User Commercial
Paradise # I23 63101A04

$753.99

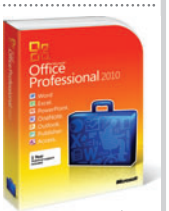
programmers.com/intel

Microsoft Office Professional 2010

by Microsoft

NEW RELEASE!

Organize projects, manage finances and build a better way to do business with tools from Microsoft® Office Professional 2010. This software suite includes 2010 versions of Word, Excel®, PowerPoint®, OneNote®, Outlook®, Publisher® and Access®. It offers a Backstage™ view which replaces the traditional File menu to give you one go-to spot to conveniently save, open and print documents. Additionally, the server integration capabilities make it easier to track. Besides this, the Office Professional Plus 2010 also offers complete package through familiar intuitive tools.



Complete DVD Windows 32/64 bit
Paradise # M47 21301A01

$442.99

programmers.com/microsoft

866-719-1528

programmersparadise.com

Prices subject to change. Not responsible for typographical errors.

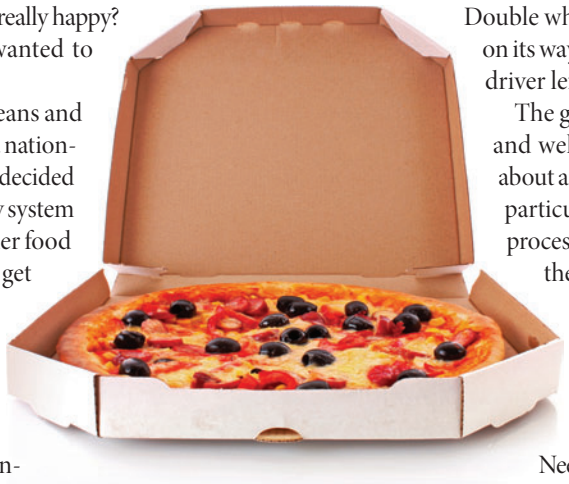
EDITOR'S NOTE

KEITH WARD

Does Your Program Smell Like Bacon?

Has a UI experience ever just made you really happy? I had one at Tech•Ed in June, and wanted to share it with you.

I was in my hotel room in New Orleans and wanted a pizza. I found the Web site of a nationally known pizza delivery company and decided to see how well its online order/delivery system worked. I'm always a bit hesitant to order food online; I'm never sure that the store will get my order, or whether the employee responsible for monitoring Internet orders will see it. Then, if they get it, will they make it right away? When will it be delivered? So many questions, so few answers from most online food-ordering systems.



> The pizza arrived, direct to my room, within 15 minutes of the driver leaving. Needless to say, I gave him a big tip.

I went through the ordering process—medium pepperoni, bacon, black olives (yeah, I'm strange). Then I hit "Order," and the magic began.

A timeline appeared—it told me that my order was received. Then it told me that the pizza was being made, and by whom (Susie). Whoa! That's something I didn't expect. Then the system informed me that the pizza was in the oven. Then out of the oven and going through a quality assurance check by Susie. Double whoa! Then it told me that the pizza was on its way, who the driver was and what time the driver left. Wowser!

The graphics were outstanding; clear, large and well-labeled. You didn't have to wonder about anything. For instance, at every stage, the particular segment on the timeline that was in process flashed; in other words, when it was in the oven, that part of the timeline flashed. I immensely enjoyed following my pizza's progress on that colorful timeline.

The pizza arrived, direct to my room, within 15 minutes of the driver leaving. Needless to say, I gave him a big tip.

This experience, to me, really demonstrated the power of customer-focused development. The UI answered any question I could possibly have about my pizza; in fact, it went above and beyond my expectations (how often does that happen?), informing me at every step of the process what was going on. As strange as it sounds, I felt personally connected to the pizza delivery company—and my pizza—through this experience. Isn't that how you want your users to feel?

The key take-away from this experience: You must do more than just "Get it to work." Think like an end user, and ask yourself, "What would make me excited about this program? What functionality would make me go 'Wow!'?" Has your program gone beyond the minimum requirements? Is the UI more than just functional—does it make users happy?

If not, you've got some work to do.

Do you have examples of great UI design? Send them to me at mmeditor@microsoft.com.

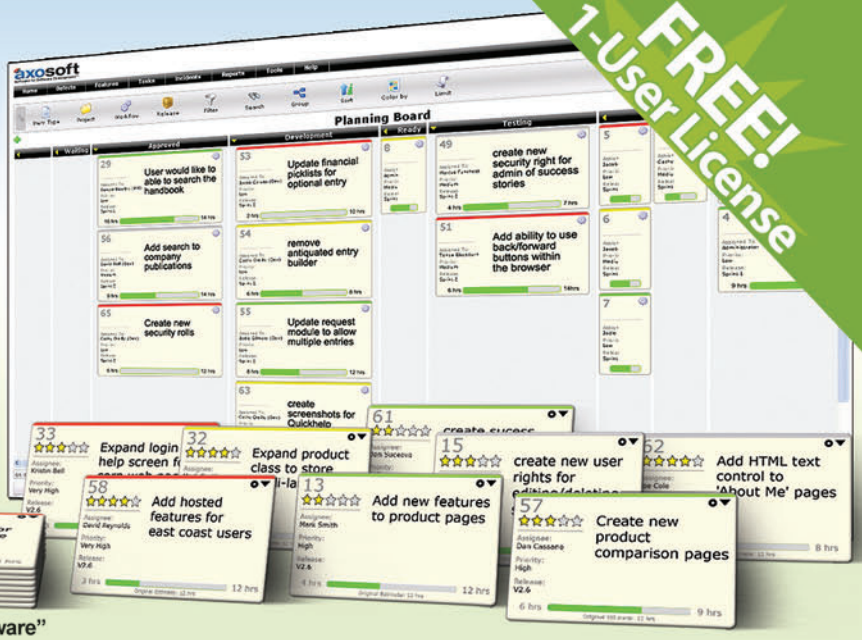
Keith Ward

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

Introducing OnTime 2010

NEW! Scrum Planning Board that Changes Everything!

FREE! 1-User License



Reader's Choice 2006 aspnet Winner · Reader's Choice 2007 aspnet Winner · Reader's Choice 2008 aspnet Winner · Reader's Choice 2009 aspnet PRO Winner

"Best Project Management Software"
Readers' Choice - 4 years in a row

Project Management • Bug Tracking • Scrum

Sign up for *OnTime Now* (in the Cloud) or install locally!

Track Everything
Track everything from bugs to software requirements, team member tasks, and more. 360-degree visibility.

Planning Board
The **NEW** planning board makes this both the most powerful & easiest to use version of OnTime **EVER!**

Tortoise SVN
Perfect for Subversion Teams! The new TortoiseSVN plug-in for OnTime 2010 provides tight integration.

Scrum
Harness the power of scrum within the OnTime client. Use Sprints, Burndowns, Trending Charts and more.

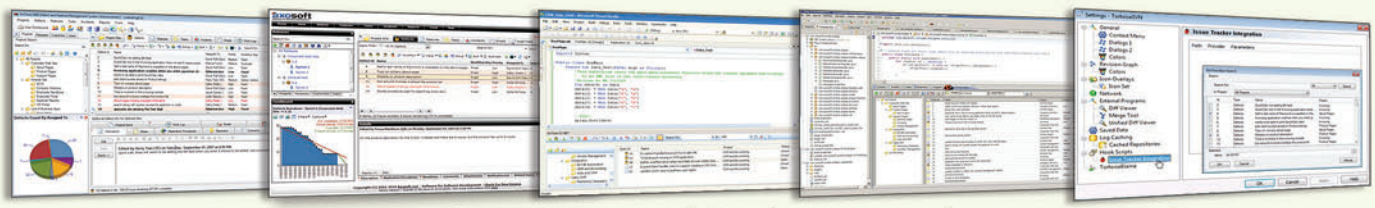
Workflow
Define a hyper-fast, lightweight, agile workflow or a robust process-enforcement system.

Help Desk
Use OnTime for tracking Help Desk incidents - even monitor email accounts to generate help tickets.

Notifications
Ensure team members are in sync with automatic email notifications and alerts.

Customize
Unlimited custom fields, customize all field names, and decide when fields are visible, editable or required.

OnTime Now!
Cloud project management done right! During signup, speed-test multiple data centers & pick the quickest.



Windows *Web* *Visual Studio* *Eclipse* *Sub-Version*

axosoft

FREE 1-User License • FREE 30-day Team Trials

800.653.0024
www.axosoft.com

DINO ESPOSITO

Don't Worry, Be Lazy

In software development, the term laziness refers to delaying certain expensive activity as long as possible much more than it relates to idleness. Software laziness is still about doing things, but it means that any action will take place only when it's needed to complete a certain task. In this regard, laziness is an important pattern in software development and can be successfully applied to a variety of scenarios, including design and implementation.

For example, one of the fundamental coding practices of the Extreme Programming methodology is summarized simply as "You Aren't Gonna Need It," which is an explicit invitation to be lazy and incorporate in the codebase only the features you need—and only when you need them.

On a different note, during the implementation of a class, you might want to be lazy when it comes to loading data from a source that isn't so cheap to access. The lazy loading pattern, in fact, illustrates the commonly accepted solution of having a member of a class defined but kept empty until its content is actually required by some other client code. Lazy loading fits perfectly in the context of object-relational mapping (ORM) tools such as the Entity Framework and NHibernate. ORM tools are used to map data structures between the object-oriented world and relational databases. In this context, lazy loading refers to a framework's ability to load, for example, Orders for a customer only when some code is trying to read the exposed Orders collection property on a Customer class.

Lazy loading, though, isn't limited to specific implementation scenarios such as ORM programming. Moreover, lazy loading is about not getting an instance of some data before it actually becomes useful. Lazy loading, in other words, is about having special factory logic that tracks what has to be created and creates it silently when the content is eventually requested.

In the Microsoft .NET Framework, we developers have long had to implement any lazy behavior manually in our classes. There's never been built-in machinery to help with this task. Not until the advent of the .NET Framework 4, that is, where we can start leveraging the new Lazy<T> class.

Meet the Lazy<T> Class

The Lazy<T> is a special factory you use as a wrapper around an object of a given type T. The Lazy<T> wrapper represents a live proxy for a class instance that doesn't yet exist. There are many reasons for using such lazy wrappers, the most important of which is improving performance. With lazy initialization of objects, you avoid any computation that isn't strictly needed, so you reduce memory consumption. If applied appropriately, lazy instantiation of objects can also be a formidable tool to make applications start much faster. The following code shows how to initialize an object in a lazy manner:

```
var container = new Lazy<DataContainer>();
```

In this example, the DataContainer class indicates a plain data container object that references arrays of other objects. Right after invoking the new operator on a Lazy<T> instance, all you have back is a live instance of the Lazy<T> class; in no case will you have an instance of the specified type T. If you need to pass an instance of DataContainer to members of other classes, you must change the signature of these members to use Lazy<DataContainer>, like this:

```
void ProcessData(Lazy<DataContainer> container);
```

> Laziness is an important pattern in software development and can be successfully applied to a variety of scenarios, including design and implementation.

When does the actual instance of DataContainer get created so that the program can work with the data it needs? Let's have a look at the public programming interface of the Lazy<T> class. The public interface is fairly slim as it includes only two properties: Value and IsValueCreated. The property Value returns the current value of the instance associated with the Lazy type, if any. The property is defined as follows:

```
public T Value
{
  get { ... }
}
```

The property IsValueCreated returns a Boolean value and indicates whether or not the Lazy type has been instantiated. Here's an excerpt from its source code:

```
public bool IsValueCreated
{
  get
  {
    return ((m_boxed != null) && (m_boxed is Boxed<T>));
  }
}
```

The m_boxed member is an internal private and volatile member of the Lazy<T> class that contains the actual instance of the T type, if any. Therefore, IsValueCreated simply checks whether a live in-

DevExpress™
Download • Compare • Decide!™

Blazing-Fast **GRID CONTROLS**
for Winforms, ASP.NET, WPF and Silverlight

WinForms Controls · ASP.NET AJAX Controls · Silverlight Controls · WPF Controls · Visual Studio Tools · Application Frameworks

FEATURE-COMPLETE PRESENTATION COMPONENTS • EASY-TO-USE REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS • BUSINESS APP FRAMEWORKS

LEARN MORE and DOWNLOAD YOUR FREE EVALUATION COPY TODAY
VISIT **DEVEXPRESS.COM/GRIDS**

Figure 1 **TheLazyThreadSafetyMode Enumeration**

Value	Description
None	The Lazy<T> instance isn't thread-safe and its behavior is undefined if it's accessed from multiple threads.
PublicationOnly	Multiple threads are allowed to concurrently try to initialize the Lazy type. The first thread to complete wins and the results generated by all others are discarded.
ExecutionAndPublication	Locks are used to ensure that only a single thread can initialize a Lazy<T> instance in a thread-safe manner.

stance of T exists and returns a Boolean answer. As mentioned, the m_boxed member is private and volatile, as shown in this snippet:

```
private volatile object m_boxed;
```

In C#, the volatile keyword indicates a member that can be modified by a concurrently running thread. The volatile keyword is used for members that are available in a multithread environment but lack (essentially for performance reasons) any protection against possible concurrent threads that could access such members at the same time. I'll return to the threading aspects of Lazy<T> later. For now, suffice it to say that public and protected members of Lazy<T> are thread-safe by default. The actual instance of the type T is created the first time any code attempts to access the Value member. The details of the object creation depend on the threading attributes optionally specified via the Lazy<T> constructor. It should be clear that implications of the threading mode are only important for when the boxed value is actually initialized or accessed for the first time.

In the Microsoft .NET Framework, we developers have long had to implement any lazy behavior manually in our classes.

In the default case, an instance of the type T is obtained via reflection by placing a call to Activator.CreateInstance. Here's a quick example of the typical interaction with the Lazy<T> type:

```
var temp = new Lazy<DataContainer>();
Console.WriteLine(temp.IsValueCreated);
Console.WriteLine(temp.Value.SomeValue);
```

Note that you are not strictly required to check IsValueCreated before invoking Value. You typically resort to checking the value of IsValueCreated only if—for whatever reason—you need to know whether a value is currently associated with the Lazy type. There's no need for you to check IsValueCreated to avoid a null reference exception on Value. The following code works just fine:

```
var temp = new Lazy<DataContainer>();
Console.WriteLine(temp.Value.SomeValue);
```

The getter of the Value property checks whether a boxed value already exists; if not, it triggers the logic that creates an instance of the wrapped type and returns that.

The Process of Instantiation

Of course, if the constructor of the Lazy type—DataContainer in the previous example—throws an exception, your code is responsible for handling that exception. The exception captured is of type TargetInvocationException—the typical exception you get when .NET reflection fails to create an instance of a type indirectly.

The Lazy<T> wrapper logic simply ensures that an instance of type T is created; in no way does it also guarantee that you won't get a null reference exception as you access any of the public members on T. For example, consider the following code snippet:

```
public class DataContainer
{
  public DataContainer()
  {
  }

  public IList<String> SomeValues { get; set; }
}
```

Imagine now that you attempt to call the following code from a client program:

```
var temp = new Lazy<DataContainer>();
Console.WriteLine(temp.Value.SomeValues.Count);
```

In this case, you'll get an exception because the SomeValues property of the DataContainer object is null, not because the DataContainer is null itself. The exception raises because the DataContainer's constructor doesn't properly initialize all of its members; the error has nothing to do with the implementation of the lazy approach.

The Value property of Lazy<T> is a read-only property, meaning that once initialized, a Lazy<T> object always returns the same instance of the type T or the same value if T is a value type. You can't modify the instance but you can access any public properties the instance may have.

Here's how you can configure a Lazy<T> object to pass ad hoc parameters to the T type:

```
temp = new Lazy<DataContainer>(() => new Orders(10));
```

One of the Lazy<T> constructors takes a delegate through which you can specify any action required to produce proper input data for the T constructor. The delegate isn't run until the Value property of the wrapped T type is accessed for the first time.

Thread-Safe Initialization

By default, Lazy<T> is thread-safe, meaning that multiple threads can access an object and all threads will receive the same instance of the T type. Let's look at aspects of threading that are important only for the first access to a Lazy object.

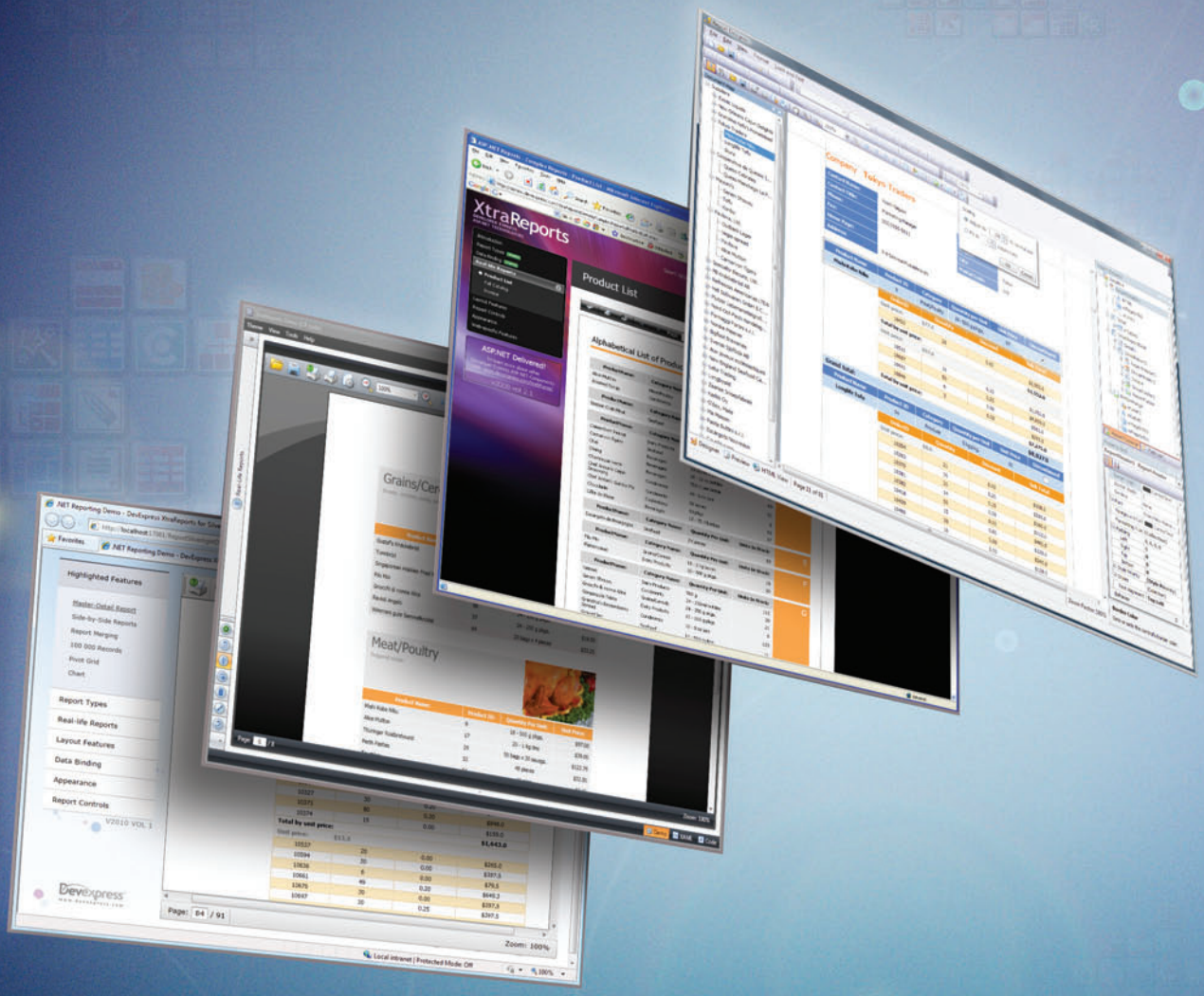
The first thread to hit the Lazy<T> object will trigger the instantiation process for type T. All following threads that gain access to Value receive the response generated by the first—whatever that is. In other words, if the first thread causes an exception when invoking the constructor of the type T, then all subsequent calls—regardless of the thread—will receive the same exception. By design, different threads can't get different responses from the same instance of Lazy<T>. This is the behavior you get when you choose the default constructor of Lazy<T>.

The Lazy<T> class, however, also features an additional constructor:

```
public Lazy(bool isThreadSafe)
```

The Boolean argument indicates whether or not you want thread safety. As mentioned, the default value is *true*, which will offer the aforementioned behavior.

DevExpress™
Download • Compare • Decide!™



Full-Featured **REPORTING**
for Winforms, ASP.NET, WPF and Silverlight

WinForms Controls · ASP.NET AJAX Controls · Silverlight Controls · WPF Controls · Visual Studio Tools · Application Frameworks

FEATURE-COMPLETE PRESENTATION COMPONENTS • EASY-TO-USE REPORTING CONTROLS
IDE PRODUCTIVITY TOOLS • BUSINESS APP FRAMEWORKS

LEARN MORE and DOWNLOAD YOUR FREE EVALUATION COPY TODAY
VISIT **DEVEXPRESS.COM/REPORTING**

Figure 2 **Example of a Lazy Property**

```
public class Customer
{
   private readonly Lazy<IList<Order>> orders;

   public Customer(String id)
   {
      orders = new Lazy<IList<Order>>( () =>
      {
         return new List<Order>();
      }
      );
   }

   public IList<Order> Orders
   {
      get
      {
         // Orders is created on first access
         return orders.Value;
      }
   }
}
```

If you pass *false* instead, the Value property will be accessed from just one thread—the one that initializes the Lazy type. The behavior is undefined if multiple threads attempt to access the Value property.

The Lazy<T> constructor that accepts a Boolean value is a special case of a more general signature where you pass the Lazy<T> constructor a value from the LazyThreadSafetyMode enumeration. **Figure 1** explains the role of each value in the enumeration.

You can set the PublicationOnly mode using either of the following constructors:

```
public Lazy(LazyThreadSafetyMode mode)
public Lazy<T>(Func<T>, LazyThreadSafetyMode mode)
```

The values in **Figure 1** other than PublicationOnly are implicitly set when you use the constructor that accepts a Boolean value:

```
public Lazy(bool isThreadSafe)
```

In that constructor, if the argument isThreadSafe is *false*, then the selected threading mode is None. If the argument isThreadSafe is set to *true*, then the threading mode is set to ExecutionAndPublication. ExecutionAndPublication is also the working mode when you choose the default constructor.

The PublicationOnly mode falls somewhere in between the full thread safety guaranteed by ExecutionAndPublication and the lack thereof you get with None. PublicationOnly allows concurrent threads to try to create the instance of the type T but ensures that only one thread is the winner. The T instance created by the winner is then shared among all other threads regardless of the instance that each may have computed.

There's an interesting difference between None and ExecutionAndPublication regarding a possible exception thrown during the initialization. When PublicationOnly is set, an exception generated during the initialization isn't cached; subsequently, each thread that attempts to read Value will have a chance to re-initialize if an instance of T isn't available. Another difference between PublicationOnly and None is that no exception is thrown in PublicationOnly mode if the constructor of T attempts to recursively access Value. That situation will raise an InvalidOperation exception when the Lazy<T> class works in None or ExecutionAndPublication modes.

Dropping thread safety gives you a raw performance benefit, but you need to be careful to prevent nasty bugs and race conditions. Thus, I recommend you use the option LazyThreadSafetyMode.None only when performance is extremely critical.

If you use LazyThreadSafetyMode.None, it remains your responsibility to ensure the Lazy<T> instance will never be initialized from more than one thread. Otherwise, you may incur unpredictable results. If an exception is thrown during the initialization, the same exception is cached and raised for each subsequent access to Value within the same thread.

ThreadLocal Initialization

By design, Lazy<T> doesn't let different threads manage their own personal instance of type T. However, if you want to allow that behavior, you must opt for a different class—the ThreadLocal<T> type. Here's how you use it:

```
var counter = new ThreadLocal<Int32>(() => 1);
```

The constructor takes a delegate and uses it to initialize the thread-local variable. Each thread holds its own data that's completely out of reach of other threads. Unlike Lazy<T>, the Value property on ThreadLocal<T> is read-write. Each access is therefore independent from the next and may produce different results, including throwing (or not) an exception. If you don't provide an action delegate via the ThreadLocal<T> constructor, the embedded object is initialized using the default value for the type—null if T is a class.

Implementing Lazy Properties

Most of the time, you use Lazy<T> for properties within your own classes, but which classes, exactly? ORM tools offer lazy loading on their own, so if you're using these tools, the data access layer probably isn't the segment of the application where you'll find likely candidate classes to host lazy properties. If you aren't using ORM tools, the data access layer is definitely a good fit for lazy properties.

Segments of the application where you use dependency injection might be another good fit for laziness. In the .NET Framework 4, the Managed Extensibility Framework (MEF) just implements extensibility and inversion of control using Lazy<T>. Even if you're not using the MEF directly, management of dependencies is a great fit for lazy properties.

Implementing a lazy property within a class doesn't require any rocket science, as **Figure 2** demonstrates.

Filling a Hole

Wrapping up, lazy loading is an abstract concept that refers to loading data only when it's really needed. Until the .NET Framework 4, developers needed to take care of developing lazy initialization logic themselves. The Lazy<T> class extends the .NET Framework programming toolkit and gives you a great chance to avoid wasteful computation by instantiating your expensive objects only when strictly needed and just a moment before their use begins. ■

Dino Esposito *is the author of "Programming ASP.NET MVC" from Microsoft Press and the coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. You can join his blog at weblogs.asp.net/despos.*

Thanks to the following technical expert for reviewing this article: *Greg Paperin*

Studio Enterprise has 8,310,799 lines of code and would cost

$710,467,541

and take **10 Years** to develop*

*Based on the Constructive Cost Model (COCOMO)
http://en.wikipedia.org/wiki/COCOMO

Do you develop for the web?

Our ASP.NET AJAX & Silverlight controls would cost

$159,009,712

Only build desktop apps?

Our WinForms & WPF controls would cost

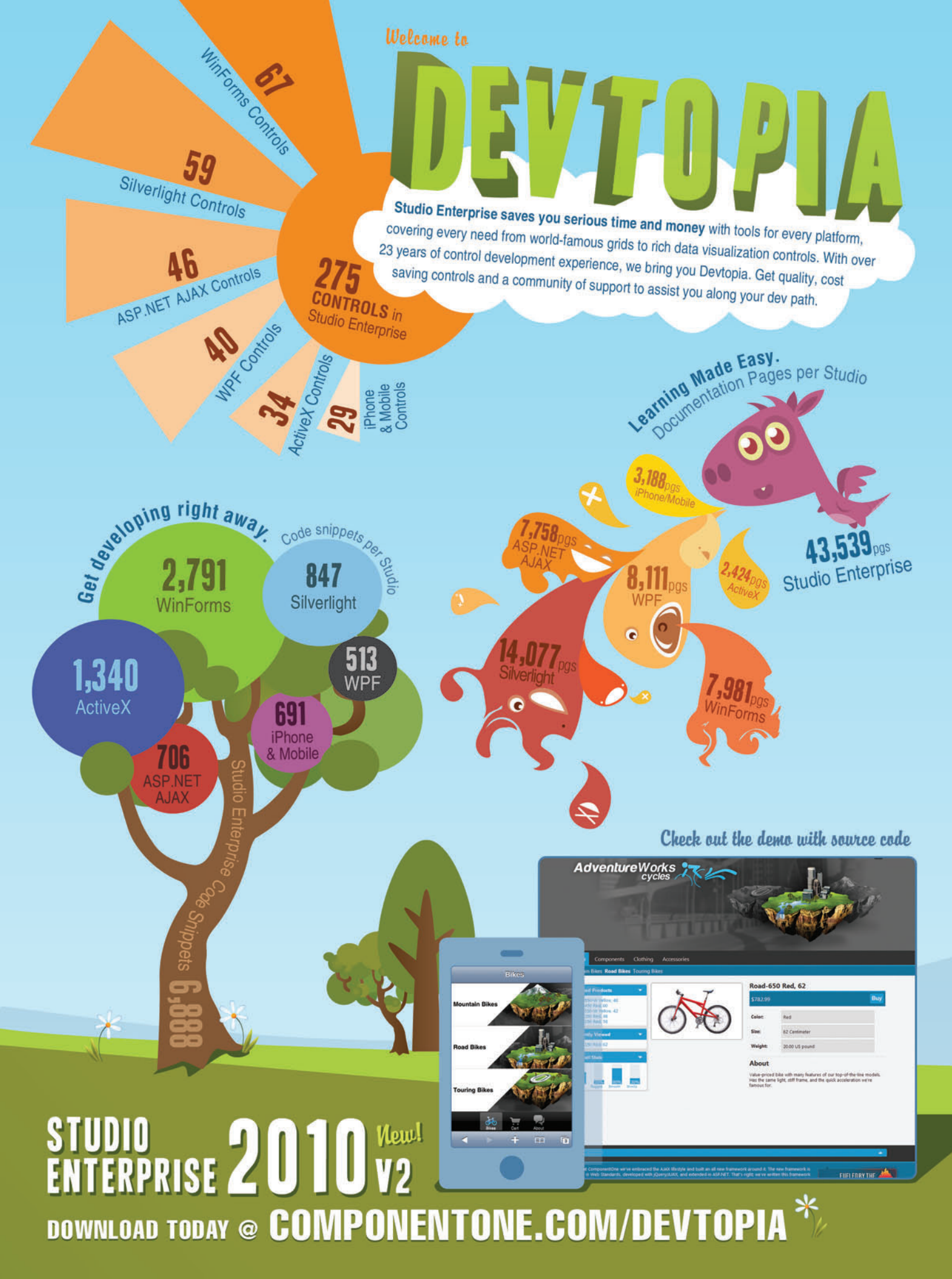
$233,189,875

Do you have mobile versions of your ASP.NET apps?

Building our Mobile Web controls would cost

$3,217,145

ComponentOne


Welcome to
DEVTOPIA
Studio Enterprise saves you serious time and money with tools for every platform, covering every need from world-famous grids to rich data visualization controls. With over 23 years of control development experience, we bring you Devtopia. Get quality, cost saving controls and a community of support to assist you along your dev path.
275 CONTROLS in Studio Enterprise
67 WinForms Controls
59 Silverlight Controls
46 ASP.NET AJAX Controls
40 WPF Controls
34 ActiveX Controls
29 iPhone & Mobile Controls
Learning Made Easy.
Documentation Pages per Studio
43,539pgs Studio Enterprise
8,111pgs WPF
7,981pgs WinForms
14,077pgs Silverlight
7,758pgs ASP.NET AJAX
3,188pgs iPhone/Mobile
2,424pgs ActiveX
Get developing right away.
Code snippets per Studio
2,791 WinForms
847 Silverlight
513 WPF
691 iPhone & Mobile
706 ASP.NET AJAX
1,340 ActiveX
Studio Enterprise Code Snippets 6,888
Check out the demo with source code
AdventureWorks cycles
Components
Clothing
Accessories
Road-650 Red, 62
Buy
About
Bikes
Mountain Bikes
Road Bikes
Touring Bikes
STUDIO ENTERPRISE 2010 New! V2
DOWNLOAD TODAY @ COMPONENTONE.COM/DEVTOPIA

JULIE LERMAN

Deny Table Access to the Entity Framework Without Causing a Mutiny

One of the first things I hear from database owners when they see the Entity Framework command creation at work is: "What? I have to provide access to my tables?" They react this way because one of the core capabilities of the Entity Framework is generation of SELECT, UPDATE, INSERT and DELETE commands.

In this column, I will give database administrators some insight into how the Entity Framework generates commands and then introduce features that let you limit its access to your database by allowing it to work only with views and stored procedures. And you can do this without impacting application code or alienating the developers on your team.

Exploring Default Command Generation

How does this command generation work? The focal point of the Entity Framework is the Entity Data Model (EDM), a conceptual model that describes an application's domain objects. The Entity Framework lets developers express queries against the model rather than concern themselves with details of the database. The model, its entities and their relationships are defined as XML, and developers work with strongly typed classes based on the model's entities. The Entity Framework runtime uses the model's XML combined with additional metadata (which describes the database schema and mappings to get from the model to the database schema) to bridge the classes with the database (see **Figure 1**).

At run time, with the help of database-specific ADO.NET providers, the Entity Framework transforms queries composed against the model into store queries—for example, T-SQL—which it then sends to the database. The Entity Framework transforms the query results into objects defined by the strongly typed entity classes as shown in **Figure 2**.

As the user works with these objects, the Entity Framework uses identity keys to track changes to properties as well as relationships between the objects. Finally, when the code calls the Entity Framework SaveChanges method to persist changes back to the database, the Entity Framework runtime reads through all of the change tracking information it has collected. For each entity that has been modified, added or deleted, the Entity Framework once again reads the model and engages the provider to build store commands and then executes them in a single, reversible transaction on the database.

This description of the default behavior of the Entity Framework tends to send database owners out of the room screaming, but I would like to emphasize the word "default" here. The Entity Framework has many default behaviors that can be altered.

How the Entity Framework processes requests to retrieve or persist data is one such behavior that can be modified. You're not required to build a model that depends on the Entity Framework to have access to your data tables. You can build a model that knows only about your database's views and stored procedures without impacting the application code that uses the model. By combining the Entity Framework's stored procedure support with its database view support, you can base all database interaction on stored procedures and views.

Mapping Entities to Database Views, Not Tables

There are a few ways to build a model. I'll focus on models that are built from a legacy database by reverse-engineering the database. Visual Studio has a wizard for this process.

In the wizard, users can select database tables, views and stored procedures. The stored procedure section also lists scalar-valued, user-defined functions that can be brought into the model.

Typically, a developer will select tables and let the wizard create entities from them. In the change-tracking and SaveChanges process I described earlier, the Entity Framework automatically generates INSERT, UPDATE and DELETE commands for the entities based on tables.

Let's first take a look at how you can force the Entity Framework to query against views instead of tables.

Database views brought into the model also become entities. The Entity Framework tracks changes to those entities just as it would for entities that are mapped to tables. There's a caveat about identity keys when using views. A database table will likely have one or more

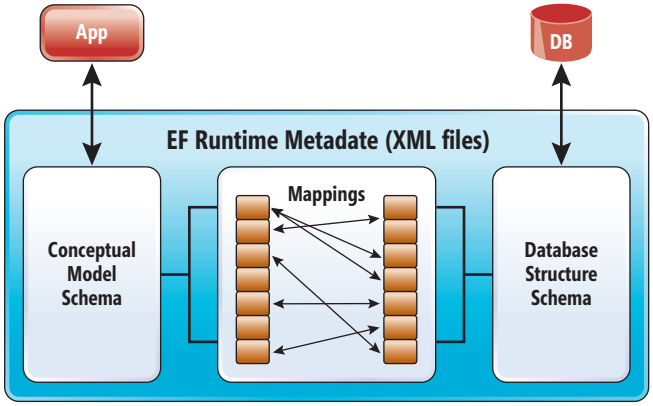


Figure 1 **The Entity Framework Runtime Metadata Is Used to Build Database Commands**

ESRI® Developer Network

Integrate Mapping and GIS into Your Applications

Give your users an effective way to visualize and analyze their data so they can make more informed decisions and solve business problems.

By subscribing to the ESRI® Developer Network (EDN℠), you have access to the complete ESRI geographic information system (GIS) software suite for developing and testing applications on every platform. Whether you're a desktop, mobile, server, or Web developer, EDN provides the tools you need to quickly and cost-effectively integrate mapping and GIS into your applications.

Subscribe to EDN and leverage the power of GIS to get more from your data. Visit www.esri.com/edn.

esri®

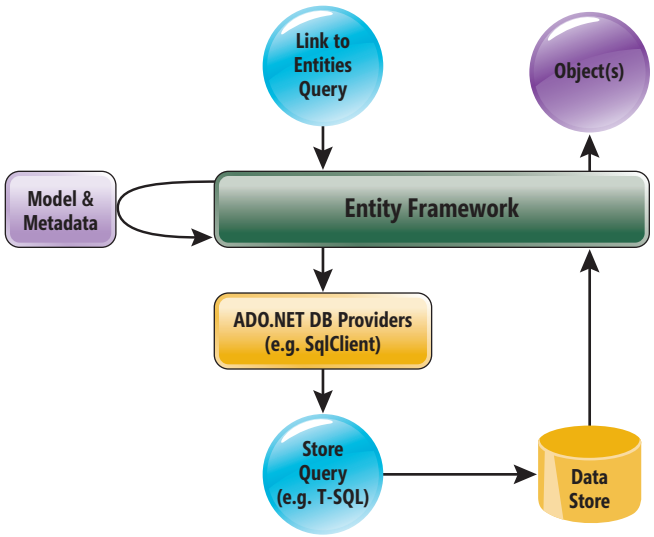


Figure 2 **The Entity Framework Executes Queries and Processes Their Results**

columns marked as its primary key or keys. By default, the wizard will compose an entity's identity key from a table's primary key(s). When creating entities that map to views (which lack primary keys), the wizard does its best job of inferring this identity key by building a composite key from all non-nullable values in the table. Consider an entity created from a view that has four non-nullable columns: ContactID, FirstName, LastName and TimeStamp.

The four resultant properties will be marked as EntityKeys (the designer uses a key icon to indicate EntityKey properties), which means that the entity has an EntityKey composed of these four properties.

The ContactID is the only property that's needed to uniquely identify this entity. Therefore, after the model has been created, you can use the designer to change the EntityKey attribute of the other three properties to False, leaving only the ContactID as a designated EntityKey.

Alternatively—if it's possible—you can plan ahead, designing database views that provide the correct, non-nullable columns.

With the key in place, the Entity Framework can uniquely identify each entity and is therefore able to perform change tracking on these entities and then persist changes back to the database when SaveChanges is called.

Overriding Command Generation with Your Own Stored Procedures

For persistence back to the database, you can override the default command generation and instead direct the Entity Framework to use your own Insert, Update and Delete stored procedures when it's time to persist changes back to the database. This is referred to as "stored procedure mapping." Let's take a look at how that works.

Any stored procedure that you select in the EDM Wizard (or subsequently in the Update Wizard) to come into your model becomes a function in the section of the model's XML metadata that describes the database schema. It isn't automatically part of the conceptual model and you won't see any representation of it on the design surface.

Here's a simple Insert stored procedure for a Person table in one of my databases.

```
ALTER procedure [dbo].[InsertPerson]
          @FirstName nchar(50),
          @LastName nchar(50),
          @Title nchar(50)
AS
INSERT INTO [Entity FrameworkWorkshop].[dbo].[Person]
          ([FirstName]
          ,[LastName]
          ,[Title]          )
     VALUES
(@FirstName,@LastName,@Title)
SELECT @@IDENTITY as PersonID
```

This stored procedure not only performs the database insert, it then returns the primary key value that SQL Server has created for the new row.

When you choose this procedure in the wizard, it's represented in the model's database schema as the following function:

```
<Function Name="InsertPerson" Aggregate="false" BuiltIn="false"
 NiladicFunction="false" IsComposable="false"
 ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
  <Parameter Name="FirstName" Type="nchar" Mode="In" />
  <Parameter Name="LastName" Type="nchar" Mode="In" />
  <Parameter Name="Title" Type="nchar" Mode="In" />
</Function>
```

You can then use the designer's Mapping Details window to map this InsertPerson function to the Person entity that was created based on the Person table, as shown in **Figure 3**.

Notice that in **Figure 3**, the PersonID property maps to the return value from the stored procedure. This particular mapping will cause the Entity Framework to update the in-memory Person object with the database-generated key once the insert has been executed in the database.

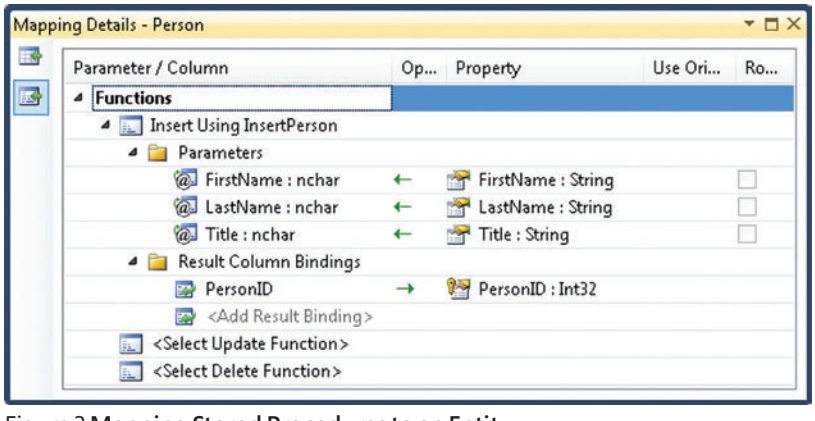


Figure 3 **Mapping Stored Procedures to an Entity**

A critical requirement when mapping functions is that every parameter in the function must map to a property in the entity. You can't map a formula or a value to the parameters. However, developers have many opportunities to customize the Microsoft .NET Framework classes that represent these entities.

You can also map the Update and Delete functions. While it isn't necessary to map all three actions (Insert, Update and Delete), developers will have to pay attention to some rules described in the documentation pertaining to mapping only some of the functions.

In **Figure 3**, notice that there are two columns to the right of property (abbreviated due to col-

LEADTOOLS®

The World Leader in Imaging Development SDKs

Silverlight, .NET, WPF, WCF, WF, C API, C++ Class Lib, COM & more!

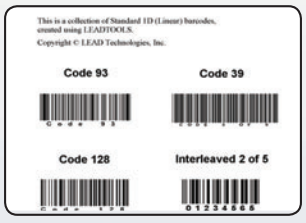
Develop your application with the same robust imaging technologies used by **Microsoft, HP, Sony, Canon, Kodak, GE, Siemens,** the **US Air Force** and **Veterans Affairs Hospitals.**

LEADTOOLS provides developers easy access to decades of expertise in color, grayscale, document, medical, vector and multimedia imaging development. Install LEADTOOLS to eliminate months of research and programming time while maintaining high levels of quality, performance and functionality.

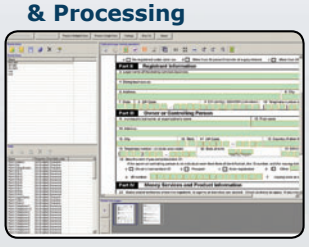
- **Image Formats & Compression:** Supports 150+ image formats and compressions including TIFF, EXIF, PDF, JPEG2000, JBIG and CCITT.
- **Display Controls:** ActiveX, COM, Win Forms, Web Forms, WPF and Silverlight.
- **Image Processing:** 200+ filters, transforms, color conversion and drawing functions supporting region of interest and extended grayscale data.
- **OCR/ICR/OMR:** Full page or zonal recognition for multithreaded 32 and 64 bit development.
- **Forms Recognition and Processing:** Automatically identify forms and extract user filled data.
- **Barcode:** Detect, read and write 1D and 2D barcodes for multithreaded 32 and 64 bit development.
- **Document Cleanup/Preprocessing:** Deskew, despeckle, hole punch, line and border removal, inverted text correction and more.
- **PDF and PDF/A:** Read and write searchable PDF with text, images and annotations.
- **Annotations:** Interactive UI for document mark-up, redaction and image measurement (including support for DICOM annotations).
- **Medical Web Viewer Framework:** Plug-in enabled framework to quickly build high-quality, full-featured, web-based medical image delivery and viewer applications.
- **Medical Image Viewer:** High level display control with built-in tools for image mark-up, window level, measurement, zoom/pan, cine, and LUT manipulation.
- **DICOM:** Full support for all IOD classes and modalities defined in the 2008 DICOM standard (including Encapsulated PDF/CDA and Raw Data).
- **PACS Communications:** Full support for DICOM messaging and secure communication enabling quick implementation of any DICOM SCU and SCP services.
- **JPIP:** Client and Server components for interactive streaming of large images and associated image data using the minimum possible bandwidth.
- **Scanning:** TWAIN 2.0 and WIA (32 and 64-bit), autodetect optimum driver settings for high speed scanning.
- **DVD:** Play, create, convert and burn DVD images.
- **DVR:** Pause, rewind and fast-forward live capture and UDP or TCP/IP streams.
- **Multimedia:** Capture, play, stream and convert MPEG, AVI, WMV, MP4, MP3, OGG, ISO, DVD and more.
- **Enterprise Development:** Includes WCF services and WF activities to create scalable, robust enterprise applications.

Multimedia

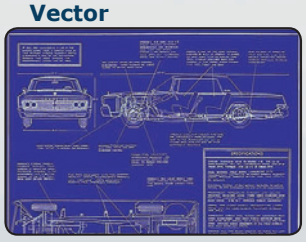
Barcode



Form Recognition & Processing

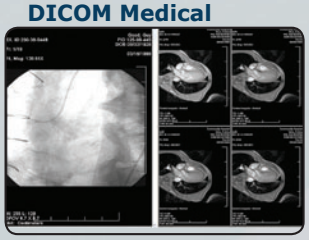


Vector



Document

DICOM Medical



High Level Design • Low Level Control

LEAD TECHNOLOGIES INCORPORATED

Free 60 Day Evaluation! **www.leadtools.com/msdn** **800 637-1840**

Figure 4 **UpdateCustomerFirstNameLastNameEmail Stored Procedure**

```
ALTER PROCEDURE UpdateCustomerFirstNameLastNameEmail
@FirstName nvarchar(50),
@LastName nvarchar(50),
@Email nvarchar(50),
@CustomerId int,
@TimeStamp timestamp

AS

UPDATE Customer
   SET [FirstName] = @FirstName
      ,[LastName] = @LastName
      ,[EmailAddress] = @Email
 WHERE CustomerID=@CustomerId AND TimeStamp=@TimeStamp

 SELECT TimeStamp
 FROM Customer
 WHERE CustomerID=@CustomerId
```

umn width): Use Original Value and Rows Affected. The Entity Framework supports optimistic concurrency, and you can use these attributes to provide concurrency checking with the Update and Delete functions. Check the MSDN document, "Walkthrough: Mapping an Entity to Stored Procedures (Entity Data Model Tools)," at msdn.microsoft.com/library/cc716679 for more information on this feature.

At run time, if a user has created a new Person type and then triggers the SaveChanges method, the Entity Framework will see the Insert function mapping in the metadata (based on the mapping defined in **Figure 3**). It will send the following command, executing the stored procedure, rather than generating its own INSERT command on the fly:

```
exec [dbo].[InsertPerson] @FirstName=N'Julie',@LastName=N'Lerman',
@Title=N'Ms.'
```

Closing the Gap and Preventing Table Access by the Entity Framework

The Entity Framework will generate commands to persist data from view-based entities, but views may not be updatable. In the case of non-updatable views, you can map Insert, Update and Delete stored procedures to the entities and get the full roundtrip of retrieving and persisting data without providing direct access to the database tables.

You could do something as simple as create database views that match the tables and create stored procedures that update the table columns. Or you might have more complex views and complex stored procedures that contain advanced logic for performing updates. You could even supplant some of your read stored procedures with views that will enable developers to compose queries over the views—something that can't be done against stored procedures.

As an example of this composability, the application could request a query against the CustomersInPastYear entity, filtering the view even further using the customer's LastName property:

```
from c in context.CustomersInPastYears
 where c.LastName.StartsWith("B")
 select c;
```

This results in the following command being executed on the database:

```
SELECT
[Extent1].[CustomerID] AS [CustomerID], [Extent1].[FirstName] AS
[FirstName],
[Extent1].[LastName] AS [LastName], [Extent1].[EmailAddress] AS
[EmailAddress],
[Extent1].[TimeStamp] AS [TimeStamp]
FROM (SELECT
      [CustomersInPastYear].[CustomerID] AS [CustomerID],
      [CustomersInPastYear].[FirstName] AS [FirstName],
      [CustomersInPastYear].[LastName] AS [LastName],
      [CustomersInPastYear].[EmailAddress] AS [EmailAddress],
      [CustomersInPastYear].[TimeStamp] AS [TimeStamp]
      FROM [dbo].[CustomersInPastYear] AS [CustomersInPastYear]) AS
[Extent1]
WHERE [Extent1].[LastName] LIKE N'B%'
```

The .NET compiler would accept a similar query composed over a stored procedure that's been mapped into the model. However, the Entity Framework would execute the stored procedure on the database, return all of its results to the application and then apply the filter to the in-memory objects returned by the stored procedure. This could potentially waste resources and hurt performance without the developer's knowledge.

Figure 4 shows a stored procedure that updates the Customer table using the same columns that participate in the CustomersInPastYear view. It can be used as the Update function for the CustomersInPastYear entity.

Now you can map this stored procedure to the entity. The mapping shown in **Figure 5** sends the original TimeStamp to the stored procedure and then, using the Result Column Bindings, captures the updated TimeStamp returned by the stored procedure.

Wrapping up, as long as the model is designed well, the view-based entities have appropriate identity keys and the functions are properly mapped, there's no need to expose your database tables to an application that uses the Entity Framework for its data access strategy. Database views and stored procedures can provide the EDM and the Entity Framework all that they need to successfully interact with your database. ■

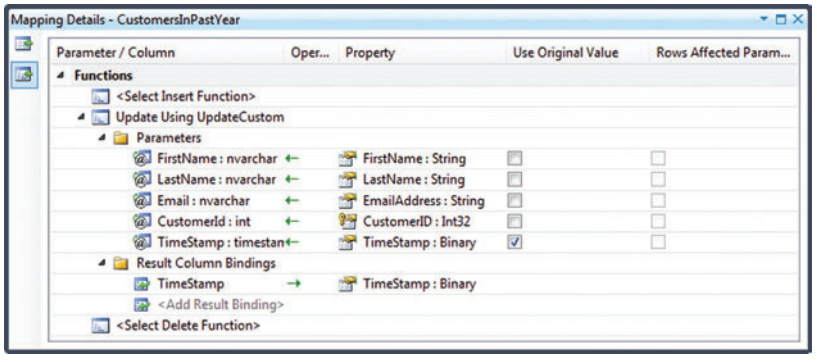


Figure 5 **Mapping a Stored Procedure to an Entity Based on a View**

JULIE LERMAN *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET topics at user groups and conferences around the world. Lerman blogs at thedatafarm.com/blog and is the author of the highly acclaimed book, "Programming Entity Framework" (O'Reilly Media, 2009). You can follow her on Twitter.com at julielerman.*

THANKS to the following technical experts for reviewing this article:
Noam Ben-Ami and Srikanth Mandadi

Now v4.0!

Why is Amyuni PDF so interesting?

Proven

Choose a PDF technology that is integrated into thousands of applications behind millions of desktops worldwide.

High-Performance

Develop with the fastest PDF conversion on the market, designed to perform in multithreaded and 64-bit Windows environments.

Rapid Integration

Integrate PDF conversion, creation and editing into your .NET and ActiveX applications with just a few lines of code.

Expertise

Produce accurate and stable PDF documents using reliable tools built by experts with over ten years of experience.

OEM Licenses

License and distribute products quickly and easily with a PDF technology that does not rely on external open-source libraries.

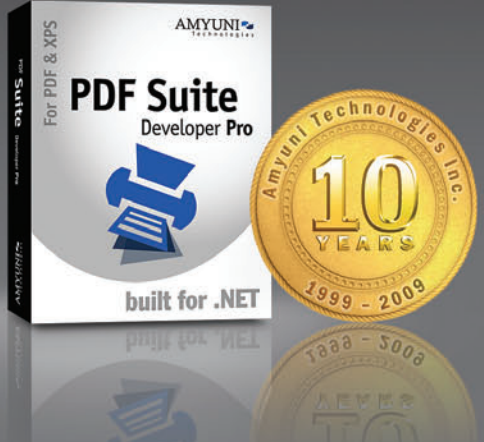
Customization

Let our experienced consultants help you turn your software requirements into customized PDF solutions.

SD Times 2009 100

Windows 7

Certified for Windows Vista



We understand the challenges that come with PDF integration. From research and development, through design and implementation, we work with you every step of the way.

Get 30 days of FREE technical support with your trial download!

www.amyuni.com

USA and Canada
Toll Free: 1 866 926 9864
Support: (514) 868 9227
Info: sales@amyuni.com

Europe
Sales: (+33) 1 30 61 07 97
Support: (+33) 1 30 61 07 98
Customizations: management@amyuni.com

AMYUNI Technologies

Passive Authentication for ASP.NET with WIF

Michele Leroux Bustamante

The goal of federated security is to provide a mechanism for establishing trust relationships between domains so that users can authenticate to their own domain while being granted access to applications and services belonging to another domain. This makes authentication techniques like single sign-on possible, removes the need to provision and manage duplicate accounts for users across applications and domains, and significantly lowers the cost to extend applications to trusted parties.

In a federated security model, an Identity Provider (IdP) performs authentication and supplies a Security Token Service (STS) to issue security tokens. These tokens, in turn, assert information about the authenticated user: her identity and possibly other information including roles and more granular access rights. In a federated world, this information is referred to as claims, and claims-based access control is central to a federated security model. In this model, applications and services authorize access to features and functionality based on claims from trusted issuers (the STS).

This article discusses:

- Passive federation basics
- WIF and passive federation
- Claims-based authorization
- Single sign-on and sign-out

Technologies discussed:

Windows Identity Foundation, ASP.NET

Platform tools like Windows Identity Foundation (WIF) make it much easier to support this type of identity federation. WIF is an identity model framework for building claims-based applications and services, and for supporting SOAP-based (active) and browser-based (passive) federation scenarios. In the article "Claims-Based Authorization with WIF," in the November 2009 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee335707), I focused on using WIF with Windows Communication Foundation (WCF). In that article I described how to implement claims-based security models for WCF services and how to migrate to identity federation.

In this follow-up article I will focus on passive federation. I will explain the flow of communication for passive federation, show you several techniques for enabling federation in your ASP.NET applications, discuss claims-based authorization techniques for ASP.NET, and talk about single sign-on and single sign-out scenarios. Along the way, I will explain the underlying WIF features and components that support passive federation scenarios.

Passive Federation Basics

Passive federation scenarios are based on the WS-Federation specification. This describes how to request security tokens and how to publish and acquire federation metadata documents, which makes establishing trust relationships easy. WS-Federation also describes single sign-on and sign-out procedures and other federation implementation concepts.

While WS-Federation discusses many details about federation, there are sections devoted to browser-based federation that

rely on HTTP GET and POST, browser redirects and cookies to accomplish the goal.

Some aspects of passive federation messaging are based closely on the WS-Trust specification. For example, passive federation employs a browser-compatible form of Request Security Token (RST) and RST Response (RSTR) when a security token is requested of an STS. In the passive federation scenario, I'll call the RST a sign-in request message and the RSTR a sign-in response message. The WS-Trust specification focuses on SOAP-based (active) federation, such as between Windows clients and WCF services.

A simple passive federation scenario is illustrated in **Figure 1**. Users authenticate to their domain and are granted access to a Web application according to their roles. The participants in this authentication scheme include the user (the subject), a Web browser (the requester), an ASP.NET application (the relying party or RP), an IdP responsible for authenticating the users within its domain and an STS belonging to the user's domain (IP-STS). A sequence of browser redirects ensures that the user is authenticated at her domain prior to accessing the RP.

The user browses to the RP application (1) and is redirected to her IdP to be authenticated (2). If the user has not yet been authenticated at the IdP, the IP-STS may present a challenge or redirect her to a login page to collect credentials (3). The user supplies her credentials (4) and is authenticated by the IP-STS (5). At this point, the IP-STS issues a security token according to the sign-in request, and the sign-in response containing the token is posted to the RP via browser redirect (6). The RP processes the security token and authorizes access based on the claims it carries (7). If successfully authorized, the user is presented with the page she originally requested and a session cookie is returned (8).

Implementing this passive federation scenario with WIF and ASP.NET involves only a few steps:

1. Establish a trust relationship between the RP and IdP (IP-STS)
2. Enable passive federation for the ASP.NET application
3. Implement authorization checks to control access to application features

In the next sections I'll discuss the features of WIF that support passive federation, walk through the steps to configure this simple scenario, and then explore other practical considerations for this and other scenarios.

WIF Features for Passive Federation

Before discussing implementation, let me review the features of WIF specifically useful for identity federation within your ASP.NET applications. To begin with, WIF supplies the following useful HTTP modules:

- **WSFederationAuthenticationModule (FAM):** Enables browser-based federation, handling redirection to the appropriate STS for authentication and token issuance, and processing the resulting sign-in response to hydrate the issued security token into a ClaimsPrincipal to be used for authorization. This module also handles other important federation messages such as sign-out requests.
- **SessionAuthenticationModule (SAM):** Manages the authenticated session by generating the session security token that contains the ClaimsPrincipal, writing it to a cookie, managing the lifetime of the session cookie and rehydrating the ClaimsPrincipal from the cookie when it's presented. This module also maintains a local session token cache.
- **ClaimsAuthorizatonModule:** Provides an extensibility point to install a custom ClaimsAuthorizationManager that can be useful for centralized access checks.
- **ClaimsPrincipalHttpModule:** Creates a ClaimsPrincipal from the current user identity attached to the request thread. In addition, provides an extensibility point to install a custom ClaimsAuthenticationManager that can be useful for customizing the ClaimsPrincipal to be attached to the request thread.

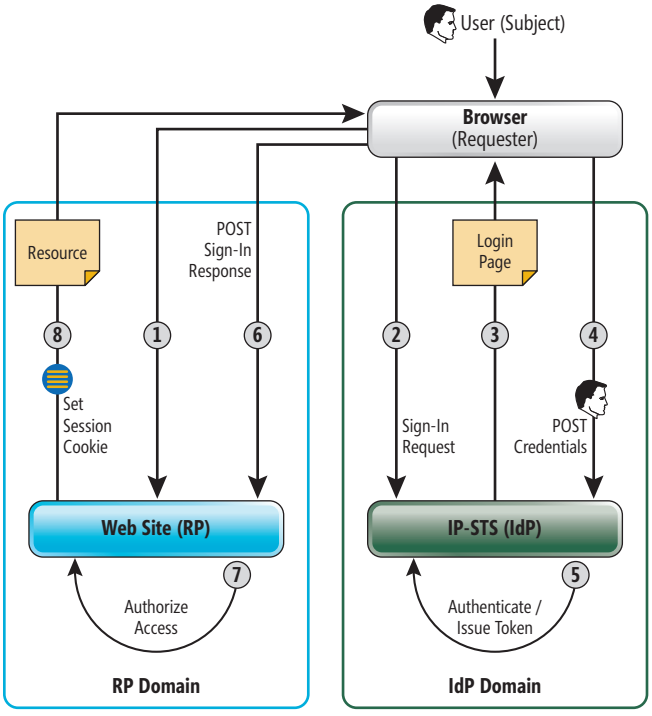


Figure 1 **A Simple Passive Federation Scenario**

Passive federation scenarios are based on the WS-Federation specification.

ClaimsPrincipalHttpModule is most useful for applications without passive federation. You can think of it as a useful tool for implementing a claims-based security model in the ASP.NET application prior to moving to passive federation. I discussed this technique for WCF in my previous article.

The other three modules are typically used together for passive federation—although ClaimsAuthorizationModule is optional. **Figure 2** illustrates how these core modules fit into the request pipeline and their functions in a typical federated authentication request.

Keeping in mind the flow of passive federation from **Figure 1,** when the user first browses to a protected page in the RP (1), access

to the application will be denied. The FAM processes unauthorized requests, produces the sign-in message and redirects the user to the IP-STS (2). The IP-STS authenticates the user (3), produces a sign-in response that includes the issued security token, and redirects back to the RP application (4).

The FAM processes the sign-in response—ensuring that the response contains a valid security token for the authenticated user—and hydrates a ClaimsPrincipal from the sign-in response (5). This will set the security principal for the request thread and HttpContext. The FAM then uses the SAM to serialize the ClaimsPrincipal to an HTTP cookie (6) that will be presented with subsequent requests during the browser session. If ClaimsAuthorizationModule is installed, it will invoke the configured ClaimsAuthorizationManager, providing an opportunity to perform global access checks (7) against the ClaimsPrincipal prior to accessing the requested resource.

Once the requested resource is presented, access control can be implemented with traditional ASP.NET login controls, IsInRole checks and other custom code that queries the user's claims (8).

On subsequent requests the session token is presented with the cookie previously written by the SAM (9). This time the SAM is engaged to validate the session token and rehydrate the ClaimsPrincipal from the token (10). The FAM is not engaged unless the request is a sign-in response, a sign-out request, or if access is denied, which can happen if the session token is not present or has expired.

In addition to these modules, there are two ASP.NET controls that are also useful in passive federation:

- **FederatedPassiveSignIn Control:** Can be used in lieu of the FAM if the application will redirect all unauthorized calls to a login page that hosts this control only when authentication is required. This assumes the user will interact with the sign-in process—useful in step-up authentication scenarios where the user is prompted for credentials, possibly additional credentials from the original login, as required by the application. The control handles redirection to the STS, processing the sign-in response, initializing the ClaimsPrincipal from the response and establishing a secure session by leveraging functionality exposed by the FAM and SAM.
- **FederatedPassiveSignInStatus Control:** This control provides an interactive way for the user to sign in or sign out from the RP application, including support for federated sign-out.

Figure 3 illustrates how the flow of communication changes when the FederatedPassiveSignIn control is employed. The application relies on Forms authentication to protect resources and redirect to the login page, which hosts the control (1). The user clicks the FederatedPassiveSignIn control (or can be redirected to it automatically), which triggers a redirect to the STS (2). The control page receives the response from the STS, relying on the FAM and the SAM to process the sign-in response (3), hydrate the ClaimsPrincipal and write the session cookie (4). When the user is redirected to the originally requested page (5), the SAM is engaged to validate the session cookie and hydrate the ClaimsPrincipal for the request. At this point, the ClaimsAuthorizationModule and that page can perform their authorization checks as illustrated in **Figure 2**.

Both the FAM and SAM rely on the appropriate SecurityTokenHandler type to process incoming tokens. When a sign-in response arrives, the FAM iterates through SecurityTokenHandlerCollection looking for the correct token handler to read the XML token. In a federated scenario this will typically be Saml11SecurityTokenHandler or Saml2SecurityTokenHandler—though other token formats may be employed if you add custom token handlers. For the SAM, SessionSecurityTokenHandler is used to process the session token associated with the session cookie.

Several identity model configuration settings are important to the flow of passive federation—and are used to initialize the FAM and the SAM and the FederatedPassiveSignIn control (although the latter also exposes properties configurable from the Visual Studio designer). Programmatically, you can supply an instance of the ServiceConfiguration type from the Microsoft.IdentityModel.Configuration namespace, or you can supply declarative configuration in the <microsoft.identityModel> section. **Figure 4** summarizes identity model settings, many of which will be discussed in subsequent sections.

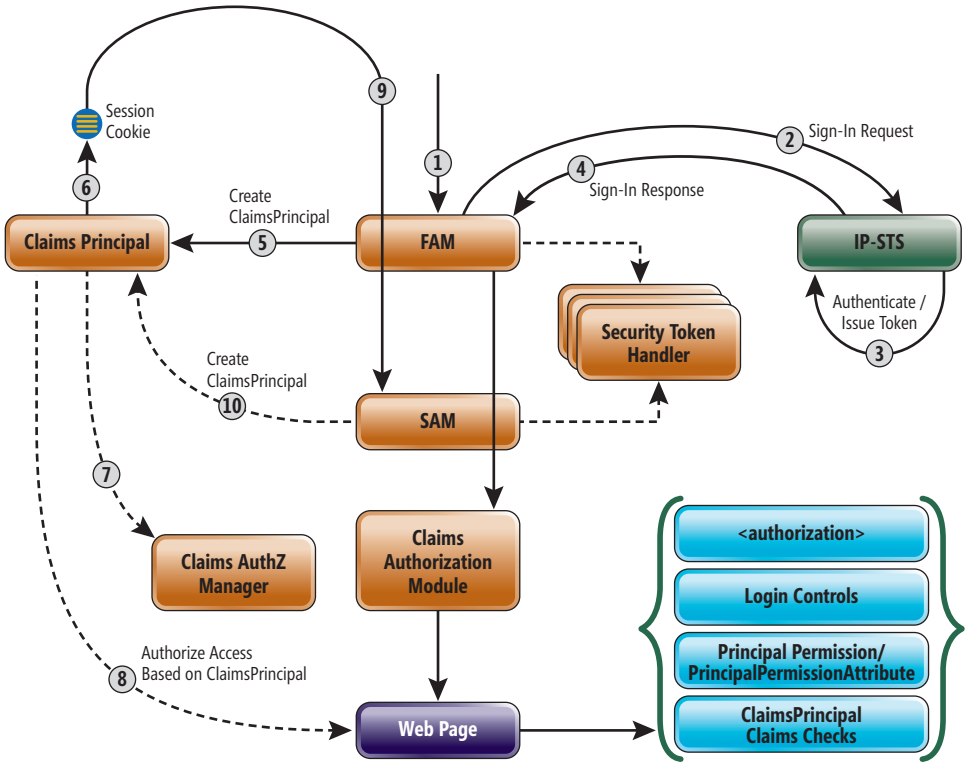
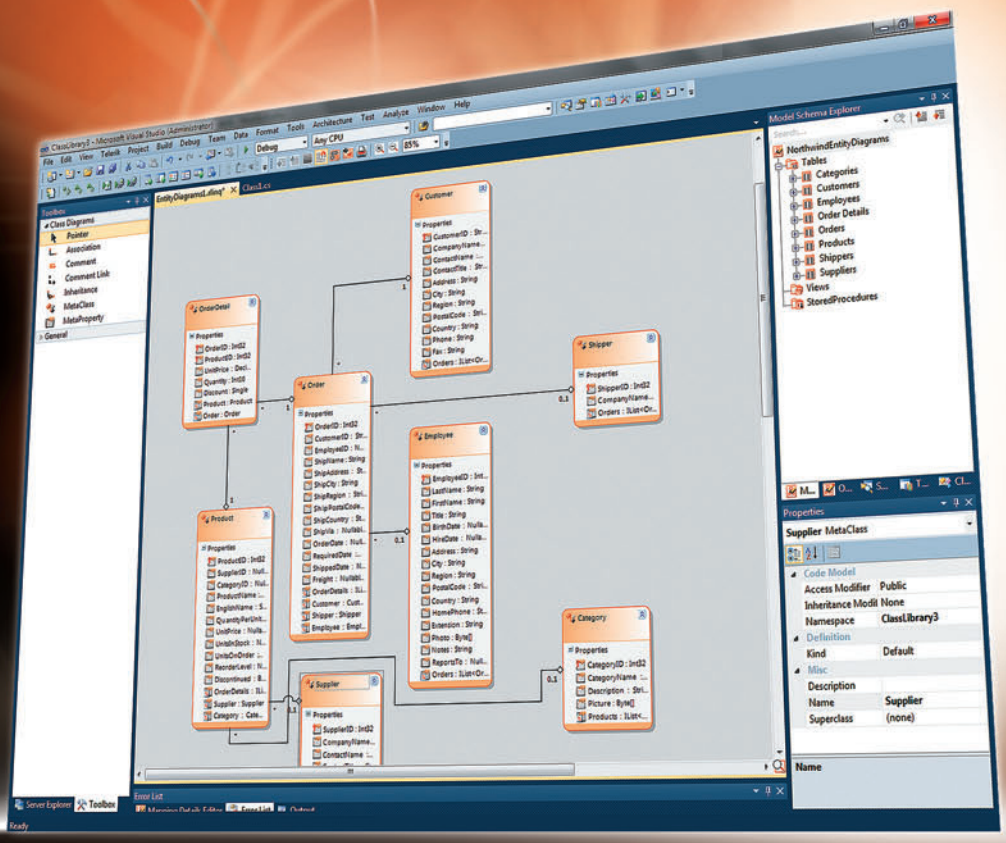


Figure 2 **WIF Components and HTTP Modules Engaged in Passive Federation**



POWERFUL DATA ACCESS.
POINT-AND-CLICK SIMPLICITY.

Presenting Telerik OpenAccess ORM – The easiest way to build your data layer

- Powerful Visual Designer
- Forward and Reverse Mapping capabilities
- Full LINQ support
- Automatic web service layer
- Silverlight support
- Optimized for complex scenarios

www.telerik.com/ORM

telerik
deliver more than expected

Europe HQ: +359.2.80.99.850 • US Sales: +1.888.365.2779 • Germany Sales: +49.89.8780687.70
e-mail: sales@telerik.com

Enabling Passive Federation

WIF makes it easy to configure passive federation for your ASP.NET applications. An STS should supply federation metadata (as described in the WS-Federation specification) and WIF supplies a Federation Utility (FedUtil.exe), which uses federation metadata to establish trust between an RP and an STS (among other features useful to both active and passive federation scenarios). You can invoke FedUtil from the command line or from Visual Studio by right-clicking on the RP project and selecting Add STS reference.

You'll complete the following simple steps with the FedUtil wizard:

- The first page of the wizard allows you to confirm the configuration file to be modified by the wizard and the RP application URI.
- The second page requests the path to the federation metadata XML document for the STS with which the RP will establish trust.
- The third page allows you to supply a certificate to be used for decrypting tokens.
- The final page shows a list of claims offered by the STS—which you can use to plan access control decisions, for example.

When the wizard steps are completed, FedUtil modifies the project to add a reference to the Microsoft.IdentityModel assembly. It also modifies the web.config to install the FAM and SAM modules and to supply identity model configuration settings for those modules. The application now supports passive federation and will redirect unauthorized requests to the trusted STS.

There's an assumption here that the STS has prior knowledge of the RP, will thus issue tokens for authenticated users trying to access the RP, and of course that it has the public key the RP requires the STS to use to encrypt tokens. This is an easy way to get your ASP.NET applications initially set up for federation. Of course, it helps to understand how to set this up from scratch in case adjustments are required, and how to go beyond the basic settings enabled by the wizard. I'll focus on the "from scratch" approach from here on in.

> WIF makes it easy to configure passive federation for your ASP.NET applications.

Without using FedUtil, you need to manually add a reference to the Microsoft.IdentityModel assembly, and manually configure the FAM and the SAM modules along with the necessary identity model settings. HTTP modules are added to two sections: system.web for Internet Information Services (IIS) 6 and system.webServer for IIS 7. Assuming the application is hosted in IIS 7, the WIF modules are configured as follows:

```
<modules>
  <!--other modules-->
  <add name="SessionAuthenticationModule"
    type="Microsoft.IdentityModel.Web.SessionAuthenticationModule, Microsoft.
IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
    preCondition="managedHandler" />
  <add name="WSFederationAuthenticationModule"
    type="Microsoft.IdentityModel.Web.WSFederationAuthenticationModule,
Microsoft.IdentityModel, Version=3.5.0.0, Culture=neutral, PublicKeyToke
n=31bf3856ad364e35"
    preCondition="managedHandler" />
</modules>
```

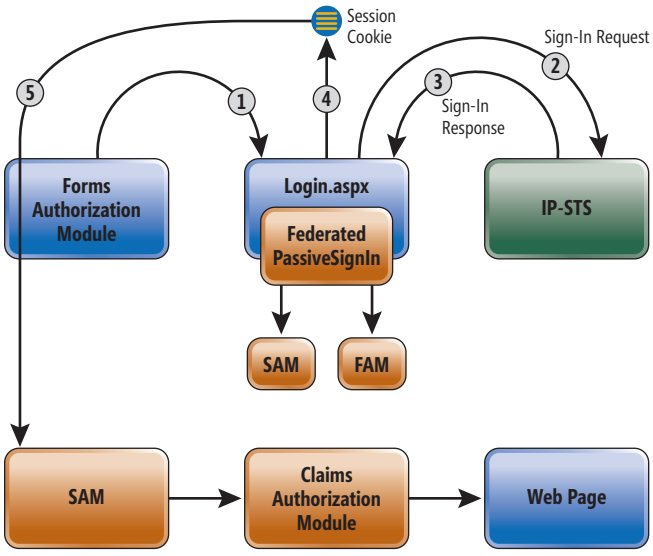


Figure 3 **Passive Federation with the FederatedPassiveSignIn Control**

By default this configuration will only protect resources with extensions explicitly mapped to be handled by the ASP.NET pipeline (.aspx, .asax, and so on). To protect additional resources with federated authentication, you should map those extensions to the ASP.NET pipeline in IIS, or you can set runAllManagedModulesForAllRequests to true in the modules setting (IIS 7 only) as follows:

```
<modules runAllManagedModulesForAllRequests="true">
```

For the FAM to kick in, you must also set the ASP.NET authentication mode to None and deny anonymous users access to application resources:

```
<authentication mode="None" />

<authorization>
  <deny users="?" />
</authorization>
```

Both modules rely on identity model configuration settings described in **Figure 4**, a typical example of which is shown in **Figure 5**. Most of these settings are generated for you by FedUtil, with the exception of certificateValidation and a few of the settings within federatedAuthentication. I typically recommend using PeerTrust certificate validation mode—which means that you explicitly add all trusted certificates, including that of the trusted issuer, to the local machine's TrustedPeople store.

You should typically require HTTPS/SSL for passive federation to protect the issued bearer token from man-in-the-middle attacks, and require HTTPS/SSL for session cookies. By default, cookies are hidden from script, but it's an important setting, which is why I call it out in **Figure 5**.

As for the name and path of the cookie, the name defaults to FedAuth, the path to the application directory. It can be useful to specify a unique name for the cookie, in particular if many RP applications in the solution share the same domain. Conversely, you can choose to specify a generic path when you want cookies to be shared across several apps on the same domain.

You will typically use FedUtil to configure your ASP.NET applications for passive federation using the FAM and SAM, then tweak the

Free Copies of #1 Rated AJAX Web DB Development tool,

(Normally the price is $395 - see below for details)

Alpha Five v10.5 with Codeless AJAX Builds Powerful & Secure Web 2.0/AJAX Database Applications 5-10x faster than other Web database development tools

Design AJAX Web apps against SQL databases that run as fast as desktop apps. Professional developers can extend Alpha Five using open Javascript libraries, the Xbasic or SQL languages, Genies and Xdialogs, the Report Builder, the Security Framework, over 1000 built-in functions, and more.

InfoWorld 2010 RAPID WEB DEVELOPMENT WINNER

eWEEK InfoWorld

InformationWeek "Alpha Five is a product we recommend watching"

TechTarget

PC MAGAZINE "Excellent"

500 no charge full copies of Alpha Five v10.5 developer have been set aside for professional developers, normally the cost on these is $395,

But if you hurry to www.alphafive.com/500free you should qualify to get one of these reserved copies.

www.alphafive.com/500free

No Purchase Required, Limited Time Offer

Figure 4 **Summary of the Essential <microsoft.identityModel> Elements**

Section	Description
<issuerNameRegistry>	Specify a list of trusted certificate issuers. This list is primarily useful for validating the token signature so that tokens signed by un-trusted certificates will be rejected.
<audienceUris>	Specify a list of valid audience URIs for incoming SAML tokens. Can be disabled to allow any URI, though not recommended.
<securityTokenHandlers>	Customize configuration settings for token handlers or supply custom token handlers to control how tokens are validated, authenticated, and serialized.
<maximumClockSkew>	Adjust the allowed time difference between tokens and application servers for token validity. The default skew is 5 minutes.
<certificateValidation>	Control how certificates are validated.
<serviceCertificate>	Supply a service certificate for decrypting incoming tokens.
<claimsAuthenticationManager>	Supply a custom ClaimsAuthenticationManager type to customize or replace the IClaimsPrincipal type to be attached to the request thread.
<claimsAuthorizationManager>	Supply a custom ClaimsAuthorizationManager type to control access to functionality from a central component.
<federatedAuthentication>	Supply settings specific to passive federation.

appropriate settings according to the requirements of the solution. You can also use the PassiveFederationSignIn control in lieu of the FAM as illustrated in **Figure 3**. The control can either load its settings from the microsoft.identityModel section, or you can set properties directly on the control.

The control approach is useful if you want unauthorized requests to be redirected to a login page where the user can explicitly sign in by clicking the control, rather than having the FAM automatically redirect to the STS. For example, if the user may belong to more than one identity provider (home realm), the login page could provide a mechanism for her to indicate her home realm prior to redirecting to the STS. I'll discuss home realm discovery shortly.

Passive Token Issuance

As mentioned earlier, passive federation relies on HTTP GET and POST and browser redirects to facilitate communication between the RP and STS. **Figure 6** shows the primary request parameters involved in the sign-in request and sign-in response during this process.

Figure 5 **Identity Model Configuration for Passive Federation**

```
<microsoft.identityModel>
  <service>
    <issuerNameRegistry type="Microsoft.IdentityModel.Tokens.
ConfigurationBasedIssuerNameRegistry, Microsoft.IdentityModel,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">
      <trustedIssuers>
        <add thumbprint="EF38A0A6D1274766093D3D78BFE4ECA77C62D5C3"
          name="http://localhost:60768/STS/" />
      </trustedIssuers>
    </issuerNameRegistry>
    <certificateValidation certificateValidationMode="PeerTrust"
      revocationMode="Online" trustedStoreLocation="LocalMachine"/>
    <audienceUris>
      <add value="http://localhost:50652/ClaimsAwareWebSite2/" />
    </audienceUris>
    <federatedAuthentication>
      <wsFederation passiveRedirectEnabled="true"
        issuer="http://localhost:60768/STS/"
        realm="http://localhost:50652/ClaimsAwareWebSite2/"
        requireHttps="true" />
      <cookieHandler requireSsl="true" name="FedAuth"
        hideFromScript="true" path="/ClaimsAwareWebSite2" />
    </federatedAuthentication>
    <serviceCertificate>
      <certificateReference x509FindType="FindByThumbprint"
        findValue="8A90354199D284FEDCBCBF1BBA81BA82F80690F2"
        storeLocation="LocalMachine" storeName="My" />
    </serviceCertificate>
  </service>
</microsoft.identityModel>
```

When the STS receives the sign-in request, it will verify that it knows about the RP by checking the wtrealm parameter against its list of known RP realms. Presumably the STS will have prior knowledge of the RP, the certificate required for token encryption, and any expectations with respect to the desired claims to be included in the issued token. The RP can indicate which claims it requires if it supplies the optional wreq parameter with a full sign-in request, and the STS can optionally respect that list or decide autonomously which claims to grant based on the authenticated user.

In a simple federation scenario like that described in **Figure 1**, there is a single RP and a single IP-STS responsible for authenticating users. If the IP-STS authenticates users against a Windows domain, it might issue role claims such as Admin, User or Guest. The assumption is that these roles have meaning to the RP for authorization. In the next section, I'll assume these roles suffice and discuss authorization techniques. Following that I will discuss claims transformation at the RP to convert STS claims into something more useful for authorization as needed.

Claims-Based Authorization

As I discussed in my previous article, role-based security in the .NET Framework expects that a security principal is attached to each thread. The security principal, based on IPrincipal, wraps the identity of the authenticated user in an IIdentity implementation. WIF supplies ClaimsPrincipal and ClaimsIdentity types based on IClaimsPrincipal and IClaimsIdentity (which ultimately derive from IPrincipal and IIdentity). When the FAM processes the sign-in response, it hydrates a ClaimsPrincipal for the issued security token. Likewise, the SAM hydrates a ClaimsPrincipal for the session cookie. This ClaimsPrincipal is the heart of WIF authorization for your ASP.NET application.

You can use any of the following approaches to authorization:

- Use location-specific authorization settings to restrict access to directories or individual application resources.
- Use ASP.NET login controls, such as the LoginView control, to control access to functionality.
- Use ClaimsPrincipal to perform dynamic IsInRole checks (for example, to dynamically hide or show UI elements).

- Use the PrincipalPermission type to perform dynamic permission demands, or the PrincipalPermissionAttribute if declarative permission demand seems appropriate on a particular method.
- Provide a custom ClaimsAuthorizationManager to centralize access checks in a single component, even prior to loading the requested resource.

The first three of these options rely on the IsInRole method exposed by the ClaimsPrincipal type. You must select a role claim type fitting for the IsInRole check so that the correct claims will be used to control access. The default role claim type for WIF is:

```
http://schemas.microsoft.com/ws/2008/06/identity/claims/role
```

If ClaimsPrincipal includes defined claims, the role claim type will match the default. Later, I will discuss permission claims in the context of claims transformation. When these are utilized, you should specify the permission claim type as the role claim type so that IsInRole will be effective.

You can control access to specific pages or directories globally from the web.config file. In the application root, supply a location tag specifying the path to protect, allow a list of acceptable roles and deny access to all other users. The following allows only Administrators to access files beneath the AdminOnly directory:

```
<location path="AdminOnly">
  <system.web>
    <authorization>
      <allow roles="Administrators" />
      <deny users="*"/>
    </authorization>
  </system.web>
</location>
```

As an alternative, you can put a web.config in any subdirectory and specify authorization rules. Placing the following configuration in the AdminOnly directory achieves the same result:

```
<configuration>
  <system.web>
    <authorization >
      <allow roles="Administrators" />
      <deny users="*"/>
    </authorization>
  </system.web>
</configuration>
```

To dynamically hide and show UI components or otherwise control access to features within a page, you can leverage the role-based features of controls such as the LoginView. However, most developers prefer to explicitly set control properties for access control during page load for more granular control. To do this, you can call the IsInRole method exposed by ClaimsPrincipal. You can access the current principal through the Thread.CurrentPrincipal static property as follows:

```
if (!Thread.CurrentPrincipal.
IsInRole("Administrators"))
  throw new SecurityException("Access is
denied.");
```

Aside from explicit IsInRole checks at runtime, you can also write classic role-based permission demands using the PrincipalPermission type. You initialize the type with the required role claim (the second constructor parameter), and when Demand is called, the IsInRole method of the current principal is called. An exception is thrown if the claim is not found:

```
PrincipalPermission p =
  new PrincipalPermission("", "Administrators");
p.Demand();
```

This approach is useful for rejecting a request with an exception, when the appropriate roles aren't present.

> Role-based security in the .NET Framework expects that a security principal is attached to each thread.

It's also useful to centralize authorization checks common to all requested resources. Sometimes, if you have an access control policy—for example, rules stored in a database—you can use a central component to read those rules to control access to features and functionality. For this, WIF supplies a ClaimsAuthorizationManager component that you can extend. Recall from my previous article that you can configure this type of custom component in the identity model section:

```
<microsoft.identityModel>
  <service>
    <!--other settings-->
    <claimsAuthorizationManager
      type="CustomClaimsAuthorizationManager"/>
  </service>
</microsoft.identityModel>
```

Figure 7 illustrates a custom ClaimsAuthorizationManager that verifies the presence of the name claim and whether the requested resource is within the AdminsOnly directory requires the Administrators role claim.

The CustomClaimsAuthorizationManager overrides CheckAccess to provide this functionality. This method supplies an AuthorizationContext parameter, which provides information about the request action (for passive federation this is an HTTP verb such as

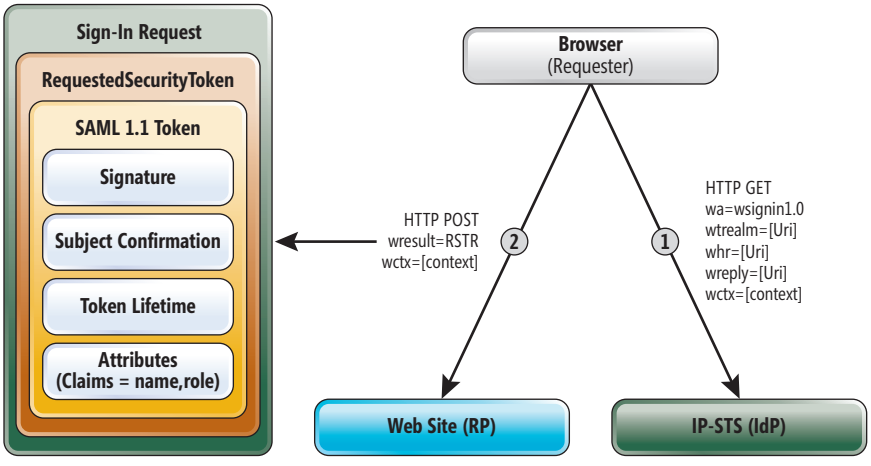
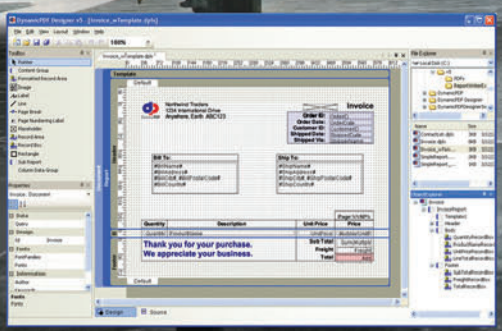


Figure 6 **Primary Sign-In Request and Response Parameters Involved in Passive Federation Requests**

Speed • Reliability • Precision • Agility

DynamicPDF…Proven .NET Components for Real-Time PDFs

DynamicPDF



Layout reports in DynamicPDF Designer with its Visual Studio look and feel.

• Easy-to-use • Highly efficient
• Industry leading support • Huge feature set

Try it FREE today!

Experience our fully functional, never expiring evaluation and community editions.

DynamicPDF Suite v6.0 for .NET

Our easy-to-use tools integrate with ASP.NET and ADO.NET allowing for the quick, real-time generation of PDF documents and reports. For easy maintenance and deployment, our most popular tools are now available as a bundled suite.

DynamicPDF Generator v6.0 for .NET

• Linearize/Fast Web View • JavaScript • Encryption/Security • PDF/X-1a • PDF/A-1a/b • Create Tagged PDFs • Interactive Form Fields • Over 50 Ready-To-Use Page Elements Including 22 Bar Codes and Charting • Digital Signatures

DynamicPDF Merger v6.0 for .NET

• Merge • Stamp • Append • Split • Password/Security • Form-Fill • Outline and Annotation Preservation • Place, Rotate, Scale and Clip Pages • Decryption

DynamicPDF ReportWriter v6.0 for .NET

• GUI Report Designer • Event Driven • Recursive Sub-Reports • Use PDF Templates • Automatic Pagination • Placeholders • Record Splitting and Expansion • Column Support • Full DynamicPDF Merger and Generator Integration

Also, check out our newest product, DynamicPDF PrintManager for .NET.

ceTe Software has been delivering quality software applications and components to our customers for over 10 years. Our DynamicPDF product line has proven our commitment to delivering innovative software components and our ability to respond to the changing needs of software developers. We back our products with a first class support team trained to provide timely, accurate and thorough responses to any support needs.

ceTesoftware
INFINITE POSSIBILITIES

www.cete.com
info@cete.com

800.631.5006
+1 410.772.8620

Figure 7 **Custom ClaimsAuthorizationManager Implementation**

```
public class CustomClaimsAuthorizationManager:
  ClaimsAuthorizationManager {

  public CustomClaimsAuthorizationManager()
  { }

  public override bool CheckAccess(
    AuthorizationContext context) {

    ClaimsIdentity claimsIdentity =
      context.Principal.Identity as ClaimsIdentity;
    if (claimsIdentity.Claims.Where(
      x => x.ClaimType == ClaimTypes.Name).Count() <= 0)
      throw new SecurityException("Access is denied.");

    IEnumerable<Claim> resourceClaims =
      context.Resource.Where(x=>x.ClaimType==ClaimTypes.Name);
    if (resourceClaims.Count() > 0) {
      foreach (Claim c in resourceClaims) {
        if (c.Value.Contains("\AdminOnly") &&
          !context.Principal.IsInRole("Administrators"))
          throw new SecurityException("Access is denied.");
      }
    }

    return true;
  }
}
```

GET or POST), the requested resource (a URI), and the ClaimsPrincipal, which is not yet attached to the request thread.

Claims Transformation

Often, the claims issued by the IP-STS, although useful for describing the authenticated user, are not relevant to the authorization requirements of the RP. It isn't the IdP's job to know what type of roles, permissions or other fine-grained artifact is necessary for authorization at each RP. It's the IdP's job to grant claims that are relevant to the identity provider domain, claims that the IdP can assert about the authenticated user.

As such, the RP may need to transform claims from the IP-STS into something more relevant for authorization. This implies that the RP may map the user identity (perhaps by user name or UPN) to a set of RP claims. Assuming the IP-STS grants default role claims, **Figure 8** lists a possible set of permission claims that the RP could issue based on each incoming role claim. The permission claim type may be a custom claim type defined by the RP such as:

```
urn:ClaimsAwareWebSite/2010/01/claims/permission
```

A good place to transform incoming IP-STS claims is with a custom ClaimsAuthenticationManager. You can install a custom ClaimsAuthenticationManager by adding the following to the microsoft.identityModel section:

```
<microsoft.identityModel>
  <service>
    <!--other settings-->
    <claimsAuthenticationManager
      type="CustomClaimsAuthenticationManager"/>
  </service>
</microsoft.identityModel>
```

Figure 9 shows a sample CustomClaimsAuthenticationManager that transforms incoming role claims granted by the IP-STS into permission claims relevant to the RP.

For IsInRole checks (as described earlier) to work, you must provide the permission claim type as the role claim type. In **Figure 9,** this is specified when the ClaimsIdentity is constructed because the RP is creating the ClaimsIdentity.

In the case where incoming SAML tokens are the source of claims, you can provide the role claims type to the SecurityTokenHandler. The following illustrates how to declaratively configure the Saml11SecurityTokenHandler to use the permission claim type as the role claim type:

```
<microsoft.identityModel>
  <service>
    <!--other settings-->
    <securityTokenHandlers>
      <remove type="Microsoft.IdentityModel.Tokens.Saml11.
Saml11SecurityTokenHandler, Microsoft.IdentityModel, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
      <add type="Microsoft.IdentityModel.Tokens.Saml11.
Saml11SecurityTokenHandler, Microsoft.IdentityModel, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35">
        <samlSecurityTokenRequirement >
          <roleClaimType
            value= "urn:ClaimsAwareWebSite/2010/01/claims/permission"/>
        </samlSecurityTokenRequirement>
      </add>
    </securityTokenHandlers>
  </service>
</microsoft.identityModel>
```

SAML token handlers have a samlSecurityTokenRequirement section where you can provide a setting for the name and role claim type, along with other settings related to certificate validation and Windows tokens.

Home Realm Discovery

So far, I have focused on a simple federation scenario with a single IP-STS. The assumption is that the RP will always redirect to a particular IP-STS to authenticate users.

In the world of federation, however, the RP may trust multiple token issuers from several domains. A new challenge presents itself in this case because the RP must decide which IP-STS should authenticate users requesting access to resources. The domain to which users authenticate is known as the user's home realm, and thus this process is called home realm discovery.

There are a number of mechanisms an application may use for home realm discovery:

- As in the current example, the home realm is known in advanced and so requests are always redirected to a particular IP-STS.
- Users may browse to the RP from another portal, which can provide a query string to indicate the home realm for users from that portal.
- The RP may require that users land on a particular entry page for each home realm. The landing page could assume a particular home realm.
- The RP may be able to determine the home realm by the IP address of the request or some other heuristic.
- If the RP can't determine the home realm from one of the aforementioned techniques, it can present a UI where the user can select the home realm or provide information that helps the RP determine this.

Figure 8 **Transforming Role Claims to Permission Claims at the RP**

Role Claim	Permission Claims
Administrators	Create, Read, Update, Delete
Users	Create, Read, Update
Guest	Read

BEST SELLER

ContourCube | from $900.00

Contour Components

OLAP component for interactive reporting and data analysis.

- Embed Business Intelligence functionality into database applications
- Zero report coding - design reports with drag and drop
- Self-service interactive reporting - get hundreds of reports by managing rows/columns
- Royalty free - only development licenses are needed
- Provides extremely fast processing of large data volumes

BEST SELLER

TX Text Control .NET and .NET Server | from $499.59

TX Text Control word processing components

Word processing components for Visual Studio .NET.

- Add professional word processing to your applications
- Royalty-free Windows Forms text box
- True WYSIWYG, nested tables, text frames, headers and footers, images, bullets, structured numbered lists, zoom, dialog boxes, section breaks, page columns
- Load, save and edit DOCX, DOC, PDF, PDF/A, RTF, HTML, TXT and XML

BEST SELLER

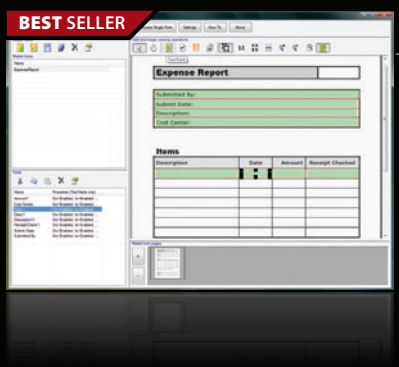
FusionCharts | from $195.02

InfoSoft Global
empowering human thoughts

Interactive and animated charts for ASP and ASP.NET apps.

- Liven up your Web applications using animated Flash charts
- Create AJAX-enabled charts that can change at client-side without invoking server requests
- Export charts as images/PDF and data as CSV for use in reporting
- Also create gauges, financial charts, Gantt charts, funnel charts and over 550 maps
- Used by over 15,000 customers and 250,000 users in 110 countries

BEST SELLER



LEADTOOLS Recognition SDK | from $3,595.50

LEAD TECHNOLOGIES

Add robust 32/64 bit document imaging & recognition functionality into your applications.

- Features accurate, high-speed multi-threaded OCR and forms recognition
- Supports text, OMR, image, and 1D/2D barcode fields
- Auto-registration and clean-up to improve recognition results
- Includes .NET, C/C++, WPF, WF, WCF and Silverlight interfaces
- Latin character set support is included. Arabic and Asian support is available.

We accept purchase orders.
Contact us to apply for a credit account.

US Headquarters
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

European Headquarters
ComponentSource
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

Asia / Pacific Headquarters
ComponentSource
3F Kojimachi Square Bldg
3-3 Kojimachi Chiyoda-ku
Tokyo
Japan
102-0083

Sales Hotline - US & Canada:
(888) 850-9911
www.componentsource.com

GSA Schedule Contract GS-35F-0188R

- If the RP supports information cards, the selected card can drive authentication to the appropriate home realm using active federation.
- The WS-Federation briefly describes how one might implement a discovery service for resolving the home realm, but there isn't a well-defined specification for this.

No matter how the home realm is discovered, the goal is to redirect the user to authenticate with the correct IP-STS. There are a few possible scenarios here. In one scenario, the RP may need to dynamically set the issuer URI so that the sign-in request is sent to the correct IP-STS. In this case, the RP must list all trusted IP-STS in the trustedIssuers section, for example:

```
<trustedIssuers>
  <add thumbprint="6b887123330ae8d26c3e2ea3bb7a489fd609a076"
    name="IP1" />
  <add thumbprint="d5bf17e2bf84cf2b35a86ea967ebab838d3d0747"
    name="IP2" />
</trustedIssuers>
```

In addition, you can override the RedirectingToIdentityProvider event exposed by the FAM and, using relevant heuristics, determine the correct URI for the STS. To do this, place the following code in the Global.asax implementation:

```
void WSFederationAuthenticationModule_RedirectingToIdentityProvider(
  object sender, RedirectingToIdentityProviderEventArgs e) {
  if (e.SignInRequestMessage.RequestUrl.Contains(
    "IP1RealmEntry.aspx")) {
    e.SignInRequestMessage.BaseUri =
      new Uri("https://localhost/IP1/STS/Default.aspx");
  }
  else if (e.SignInRequestMessage.RequestUrl.Contains(
    "IP2RealmEntry.aspx")) {
    e.SignInRequestMessage.BaseUri = new Uri(
      "https://localhost/IP2/STS/Default.aspx");
  }
}
```

The other scenario involves passing the home realm parameter (whr) with the sign-in request to the primary STS. The RP may, for example, have a Resource STS (R-STS or RP-STS) responsible for claims transformation. The RP-STS doesn't authenticate users (it's not an IdP), but it has trust relationships with one or more other IdPs.

> The domain to which users authenticate is known as the user's home realm.

The RP has a trust relationship with the RP-STS, and will always respect tokens issued by the RP-STS. The RP-STS is responsible for redirecting to the correct IdP for each request. The RP-STS can determine the correct IP-STS to redirect to as in the code just described, but another option is for the RP to supply information about the home realm, passing this in the home realm parameter to the RP-STS. In this case, the RP dynamically sets the home realm parameter:

```
void WSFederationAuthenticationModule_RedirectingToIdentityProvider(
  object sender, RedirectingToIdentityProviderEventArgs e) {
  if (e.SignInRequestMessage.RequestUrl.Contains(
    "IP1RealmEntry.aspx")) {
    e.SignInRequestMessage.HomeRealm =
      "https://localhost/IP1/STS/Default.aspx";
  }
  else if (e.SignInRequestMessage.RequestUrl.Contains(
    "IP2RealmEntry.aspx")) {
    e.SignInRequestMessage.HomeRealm =
      "https://localhost/IP2/STS/Default.aspx";
  }
}
```

The RP-STS uses this parameter to redirect to the correct IP-STS and subsequently transforms claims from the IP-STS into claims relevant to the RP.

Figure 9 **Custom Claims Transformation at the RP**

```
public class CustomClaimsAuthenticationManager:
  ClaimsAuthenticationManager {

  public CustomClaimsAuthenticationManager() { }

  public override IClaimsPrincipal Authenticate(
    string resourceName, IClaimsPrincipal incomingPrincipal) {

    IClaimsPrincipal cp = incomingPrincipal;
    ClaimsIdentityCollection claimsIds =
      new ClaimsIdentityCollection();

    if (incomingPrincipal != null &&
      incomingPrincipal.Identity.IsAuthenticated == true) {

      ClaimsIdentity newClaimsId = new ClaimsIdentity(
        "CustomClaimsAuthenticationManager", ClaimTypes.Name,
        "urn:ClaimsAwareWebSite/2010/01/claims/permission");

      ClaimsIdentity claimsId =
        incomingPrincipal.Identity as ClaimsIdentity;
      foreach (Claim c in claimsId.Claims)
        newClaimsId.Claims.Add(new Claim(
          c.ClaimType, c.Value, c.ValueType,
          "CustomClaimsAuthenticationManager", c.Issuer));

      if (incomingPrincipal.IsInRole("Administrators")) {
        newClaimsId.Claims.Add(new Claim(
          "urn:ClaimsAwareWebSite/2010/01/claims/permission",
          "Create"));
        newClaimsId.Claims.Add(new Claim(
          "urn:ClaimsAwareWebSite/2010/01/claims/permission",
          "Read"));
        newClaimsId.Claims.Add(new Claim(
          "urn:ClaimsAwareWebSite/2010/01/claims/permission",
          "Update"));
        newClaimsId.Claims.Add(new Claim(
          "urn:ClaimsAwareWebSite/2010/01/claims/permission",
          "Delete"));
      }

      else if (incomingPrincipal.IsInRole("Users")) {
        newClaimsId.Claims.Add(new Claim(
          "urn:ClaimsAwareWebSite/2010/01/claims/permission",
          "Create"));
        newClaimsId.Claims.Add(new Claim(
          "urn:ClaimsAwareWebSite/2010/01/claims/permission",
          "Read"));
        newClaimsId.Claims.Add(new Claim(
          "urn:ClaimsAwareWebSite/2010/01/claims/permission",
          "Update"));
      }

      else {
        newClaimsId.Claims.Add(new Claim(
          "urn:ClaimsAwareWebSite/2010/01/claims/permission",
          "Read"));
      }

      claimsIds.Add(newClaimsId);
      cp = new ClaimsPrincipal(claimsIds);
    }

    return cp;
  }
}
```

WE ARE

SPREADSHEETS

Act now and save up to $300!

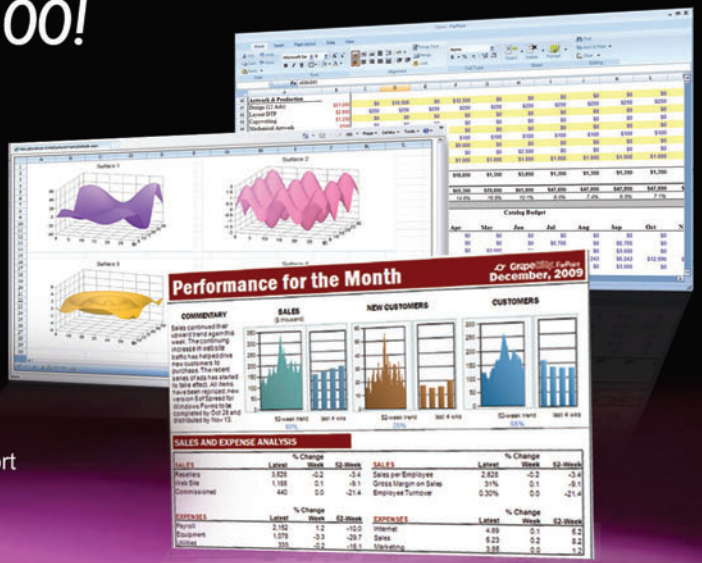
GrapeCity PowerTools

SPREAD 5

Windows Forms & ASP.NET

Award-winning Microsoft® Excel® compatible spreadsheet components for .NET and ASP.NET

- World's best-selling .NET spreadsheet technology
- Hundreds of Chart styles for data visualization
- Full featured Formula support, including most Excel functions
- Full support for native Microsoft Excel files and data import/export
- Spreadsheet Designers, Quick-start Wizard and Chart Wizards



GCPowerTools.com/ActNow

WE ARE

GRAPECITY

Excel Report Analyze

1-800-645-5913 / 1-919-460-4551

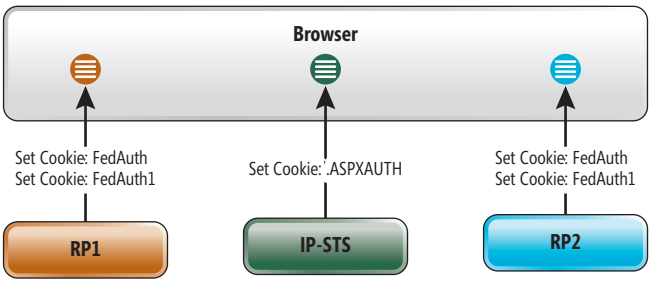


Figure 10 **Session Cookies Associated with Each RP and the STS**

Single Sign-On and Single Sign-Out

Single sign-on and single sign-out are important parts of federation. Single sign-on is a feature that allows authenticated users to access multiple RP applications while authenticating only once. Single sign-out, as it implies, facilitates sign-out from all RP applications and any relevant STS chain with a single request.

In a simple federation scenario like that shown in **Figure 1**, the user authenticates to the IP-STS and is authorized at the RP based on the issued security token. Post-authentication, the user has a session cookie for the STS and another for the RP. Now, if the user browses to another RP, she will be redirected to the IP-STS for authentication—assuming both RP applications trust the same IP-STS. Because the user already has a session with the IP-STS, the STS will issue a token for the second RP without prompting for credentials. The user now has access to the second RP and has a new session cookie for the second RP.

As I've discussed, WIF supplies the SAM to write out the session cookie for authenticated users. By default, this session cookie is issued for the relative application address for the domain, and its base name is FedAuth. Because federated session cookies can be large, the token is usually split into two (or more) cookies: FedAuth, FedAuth1, and so on.

If you are hosting more than one application at the same domain, as part of the federation scenario, the default behavior would be that the browser has a FedAuth cookie for each RP (see **Figure 10**). The browser sends only those cookies associated with the domain and path for the request.

This default behavior is generally fine, but sometimes it's necessary to supply a unique, per-application name for each session cookie—in particular if they're hosted on the same domain. Or multiple applications at the same domain may share a session cookie, in which case you can set the cookie path to "/".

If the session cookie expires, the browser will remove it from the cache and the user will be redirected once again to the STS for authentication. Separately, if the issued token associated with the session cookie has expired, WIF will redirect to the STS for a new token.

Sign-out is more explicit—usually driven by the user. Single sign-out is an optional feature of the WS-Federation specification that suggests the STS should also notify other RP applications for which it has issued tokens of the sign-out request. This way, the session cookie is removed for all applications the user browsed to during the single sign-on session. In a more complex scenario, where multiple STSs are involved, the primary STS receiving the sign-out request should also notify other STSs to do the same.

For the purpose of this discussion, I will focus on what you should do at the RP to facilitate federated single sign-out. You can place the FederatedPassiveSignInStatus control on any page from which you want to support sign-in and sign-out and the control will automatically indicate its state. Once signed-in, the control presents a link, button or image for signing out.

When you click the control, it will handle sign-out according to the SignOutAction property, which can be Refresh, Redirect, RedirectToLoginPage or FederatedPassiveSignOut. The first three delete the session cookie for the application, but do not notify the STS of the sign-out request. When you select the FederatedPassiveSignOut setting, the control will call SignOut on WSFederationAuthenticationModule. This ensures that federated session cookies are removed for the application. In addition, a sign-out request is sent to the STS:

```
GET https://localhost/IP1/STS?wa=wsignout1.0
```

If you aren't using the FederatedPassiveSignInStatus control, you can directly call WSFederationAuthenticationModule.SignOut to trigger a redirect to the STS with the sign-out request.

Single sign-out implies that the user is signed out of all applications she signed into with her federated identity. If the STS supports this, it should hold a list of RP applications the user logged in to during her session, and issue a clean-up request to each RP when federated sign-out is requested:

```
GET https://localhost/ClaimsAwareWebSite?wa=wsignoutcleanup1.0
```

> Single sign-on and single sign-out are important parts of federation.

In more complex scenarios, the same clean-up request should be sent to any other STS involved in the federated session. To that end, the STS would have to have prior knowledge of the clean-up URI for each RP and STS. To support single sign-out, your RPs should be able to process these clean-up requests. Both the FAM and the FederatedPassiveSignInStatus control support this. If you're using the FAM, the clean-up request can be posted to any URI at the RP and the FAM will process the request and clean up any session cookies. If you're using the FederatedPassiveSignInStatus control, the clean-up request must be posted to a page that contains the control.

In fact, the WS-Federation specification does not detail how to implement single sign-out and clean-up behavior beyond the recommended query strings and flow of communication. It's not easy to guarantee single sign-out will be effective across all federation partners as a result—but if you own the environment and want to achieve this goal, it's indeed possible. ■

Michele Leroux Bustamante *is chief architect at IDesign (idesign.net) and chief security architect at BiTKOO (bitkoo.com). She's also a Microsoft regional director for San Diego and a Microsoft MVP for Connected Systems. Visit her blog at micheleleroux bustamante.com.*

Thanks to the following technical expert for reviewing this article: *Govind Ramanathan*

Tips for Migrating Your Applications to the Cloud

George Huey and Wade Wegner

One of our favorite aspects of technology is that it is constantly evolving and continually changing—there's always more to learn! As students and followers of cloud computing, we're tremendously excited about the Windows Azure platform. As technical evangelists for Microsoft, we have the great fortune to work with customers in the adoption of new technology. As a result, we've seen a host of different ways in which to apply Windows Azure.

Early on, George had a personal reason for wanting to use Windows Azure. George is involved in many community activities, and the ability to quickly spin up temporary applications and spin them down when no longer needed proved tremendously useful. For developers with experience writing Microsoft .NET Framework code, there's hardly any learning curve—build your application, deploy it and run it.

Because of the interest many of our corporate customers expressed in Windows Azure, we decided to hold a set of Windows Azure Migration Labs at the Microsoft Technology Centers. The intent was for customers to bring their applications into the lab and actually migrate them to Windows Azure. Through this process, every single customer was able to successfully migrate its Web applications and SQL databases to the Windows Azure platform.

We weren't surprised—we already had plenty of experience with Windows Azure and were confident our customers would meet with success. But in the course of helping the lab attendees migrate their various applications, we learned quite a few tricks that help migrations go smoothly. In this article, we'll share some of those tips and tricks we discovered working with customers on real-world migrations.

This article discusses:

- Planning for security
- Application compatibility
- Database compatibility
- Migrating your data to SQL Azure

Technologies discussed:

Windows Azure, SQL Azure, SQL Server

Migration Basics

When deciding to migrate an application from on-premises to the cloud (or to create a new application on a cloud service), there are several aspects of the application architecture that need to be considered:

- Application management
- Application security
- Application compatibility
- Database compatibility

The questions and concerns we heard most frequently during the migration labs tended to revolve around these four areas. As a result, we'll focus our discussion around these topics.

One misconception we often encountered was the idea that, by using Windows Azure, developers don't have to worry about common

architectural patterns regarding issues such as availability, scalability, reliability and security when moving to or creating applications in the cloud. The truth is that architectural patterns in the context of distributed computing are equally valid for applications architected for on-premises deployment or Windows Azure deployment.

Application Management

No matter whether your application is running on-premises or in the cloud, the operations management team needs data that will enable them to make effective decisions. The issues you'll need to consider include service-level agreements, capacity planning, customer billing, auditing, application monitoring, traffic analysis and managing costs (knowing when to scale up or down). These need to be resolved before the application is deployed to production—and for best results, often before the application is created.

These were just some of the issues that were considered during the Windows Azure Migration Labs. By utilizing the Windows Azure Diagnostics API provided in the Windows Azure SDK (Microsoft.WindowsAzure.Diagnostics), customers were able to expose application crash dumps, failed request tracing, Windows event logs, IIS logs, Windows Azure logs and performance counters.

This is much more straightforward than you might expect. You tell the diagnostic monitor which types of diagnostic information to collect (see **Figure 1** for an example) and set the data transfer schedule for the information to be transferred to a central Windows Azure storage location.

For more information about Windows Azure diagnostics, see the article "Cloud Diagnostics: Take Control of Logging and Tracing in Windows Azure" by Mike Kelley, in the June 2010 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ff714589).

Application Security

A top concern of any organization moving to the cloud is security. Most companies have invested a substantial amount of time, money and engineering into designing and developing a security model and it's important that they're able to leverage existing investments such as identity stores, single sign-on solutions and firewalls.

While there are many ways for a company to go about securing cloud-based applications, an increasingly popular pattern is a claims-based approach.

This process is shown in **Figure 2**. In order for an application to be able to process security tokens from a Security Token Service (STS), a trust relationship must be established between the STS and the application.

The access control rules (step 1) are defined to meet business requirements (who can log into the application). These rules are stored with the STS. When a user tries to access the application, she's redirected to the STS so she can receive a valid token (step 2). The user provides a set of input claims (for example, a Live ID or a domain account) to the STS for authentication purposes. The STS will map these claims to a set of output claims once the user is authenticated (step 3). In step 4, the output claims are packaged into a Security Assertions Markup Language (SAML) token, signed by the STS, and returned to the user for forwarding to the application (the relying partner in step 5). The application confirms that the SAML token is valid and from the trusted STS (step 6). Once the token is validated, the application checks the claims in the token and sends back the appropriate response (step 7). Pretty simple! The beauty of this approach is that it fits in extremely well within the ASP.NET provider model. The process of making your ASP.NET application claims-aware is really quite simple.

In order to make life easier for the developer, Microsoft introduced the Windows Identity Foundation (WIF) SDK. This does all of the grunt work of parsing SAML 2.0 tokens, letting the developer concentrate on his application without having to worry about the underlying security technology.

The first thing to do is download WIF (microsoft.com/downloads/details.aspx?FamilyID=eb9c345f-e830-40b8-a5fe-ae7a864c4d76) and the WIF SDK (microsoft.com/downloads/details.aspx?familyid=C148B2DF-C7AF-46BB-9162-2C9422208504). Once these are installed, you'll have what you need to make your application claims-aware.

In the Visual Studio solution with your ASP.NET Web application, right-click and select Add | Add New Web Site. Select the

Figure 1 **Setting Up Diagnostics**

```
public class WebRole : RoleEntryPoint {
  public override bool OnStart() {
    DiagnosticMonitorConfiguration config =
      DiagnosticMonitor.GetDefaultInitialConfiguration();

    // To see which counters you can capture, type
    // "typeperf.exe /q" in a command window.

    // Capture CPU utilization.
    PerformanceCounterConfiguration procUtilization =
      new PerformanceCounterConfiguration();
    procUtilization.CounterSpecifier =
      @"Processor(*)\% Processor Time";
    procUtilization.SampleRate =
      System.TimeSpan.FromSeconds(30.0);
    config.PerformanceCounters.DataSources.Add(procUtilization);

    // Monitor available memory.
    PerformanceCounterConfiguration procAvailMemory =
      new PerformanceCounterConfiguration();
    procAvailMemory.CounterSpecifier = @"\Memory\Avail MBytes";
    procAvailMemory.SampleRate =
      System.TimeSpan.FromSeconds(30.0);
    config.PerformanceCounters.DataSources.Add(procAvailMemory);

    // Add event collection from Windows Event Log
    // (System and Application event logs).
    config.WindowsEventLog.DataSources.Add("System!*");
    config.WindowsEventLog.DataSources.Add("Application!*");

    // All of the information monitored so far is being stored locally.
    // Tell diagnostic monitor what schedule period should be used when
    // transfering the events.
    config.Directories.ScheduledTransferPeriod =
      TimeSpan.FromMinutes(1);
    config.Logs.ScheduledTransferPeriod =
      TimeSpan.FromMinutes(1);

    // Start the diagnostics monitor.
    DiagnosticMonitor.Start("DiagnosticsConnectionString", config);

    // True gives full crash dumps. False gives small crash dumps.
    CrashDumps.EnableCollection(false);

    System.Diagnostics.Trace.TraceInformation("OnStart Completed");

    RoleEnvironment.Changing += RoleEnvironmentChanging;

    return base.OnStart();
  }
...
```

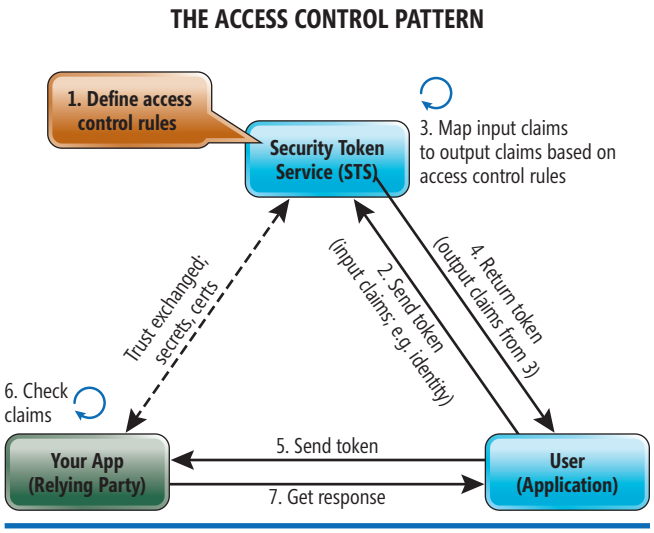


Figure 2 **Claims-Based Identity in an Application Context**

ASP.NET Security Token Service Web Site template. This will allow you to set up an STS for your development environment.

Once you have your STS created, you can add a reference to the STS by right-clicking on your application and clicking "Add STS reference." This starts a wizard that walks you through the process of establishing a relationship between your application and the STS. Point to the application's web.config file for your site and specify the Application URI (see **Figure 3**).

In the next step, choose "Use an existing STS" and then specify the location of the FederationMetadata.xml file in the STS project (see **Figure 4**). Choose the defaults for the remainder of the process.

Take a look at your web.config file. You'll see that the FedUtil.exe wizard changed a substantial amount of code. The most important changes were made to the microsoft.identityModel node of the web.config file. Here you'll see references to your STS project, along with the claim types expected by the application. To ensure that your application is appropriately receiving the claims back from your STS, put the following code in your default.aspx page (note that you'll have to add a reference to the Microsoft.IdentityModel from the WIF SDK):

```
IClaimsIdentity ici =
  (IClaimsIdentity)Thread.CurrentPrincipal.Identity;

foreach (Claim c in ici.Claims) {
  Response.Write(c.ClaimType + " - " + c.Value + "<br/>");
}
```

When you next run your application, you will be automatically redirected to your STS. By default, the STS allows you to authenticate as "Adam Carter." Just click the Login button (you don't need a password).

After the STS authenticates the login, you will be redirected back to your Web application, along with a SAML token needed for authentication. Your application will accept the token and allow the default.aspx page to run. Because the WIF modules intercept your security credentials, you're able to cast the identity principal as an IClaimsIdentity, and, consequently, you can extract the claims type and value out of the identity object (see **Figure 5**).

Now that the Web application is claims-aware, it's easy to adapt it to your existing identity model. Simply update your configuration file so that it points to your production STS and ensure that you've configured your application as a relying party. Additionally, you can use this information to create a custom role provider so that you can translate claims types into roles.

This is an extremely powerful technique, and will let you move your applications to almost any environment—on-premises, the cloud or even a partner datacenter—and still validate against your identity stores through a publicly exposed STS.

Application Compatibility

Windows Azure is an application platform, so it's important to understand the types of applications suited to the Windows Azure platform. While you have the ability to run native code and you can run applications with full trust, you must package your application before deploying it to the cloud, which means it's important to evaluate your application to see if it's a good fit.

Here's an example. One of our customers at the Windows Azure Migration Labs had an existing application consisting of a SQL Server 2005 back end, a LINQ to SQL data-access layer and a front end using both MVC Framework 1.0 and ASP.NET 3.5 SP1 running on IIS.

The application sat in a Web farm with a load balancer routing the traffic. The application itself was stateless so it didn't matter to which server the user was ultimately directed.

An interesting detail about this application was that the MVC application manages more than 220 separate Web sites. The company used a combination of MVC routing and information stored in the SQL Server database to determine which content should be loaded for each Web site. There were five Web servers behind the load balancer serving more than 4 million page visits per month for the collection of Web sites.

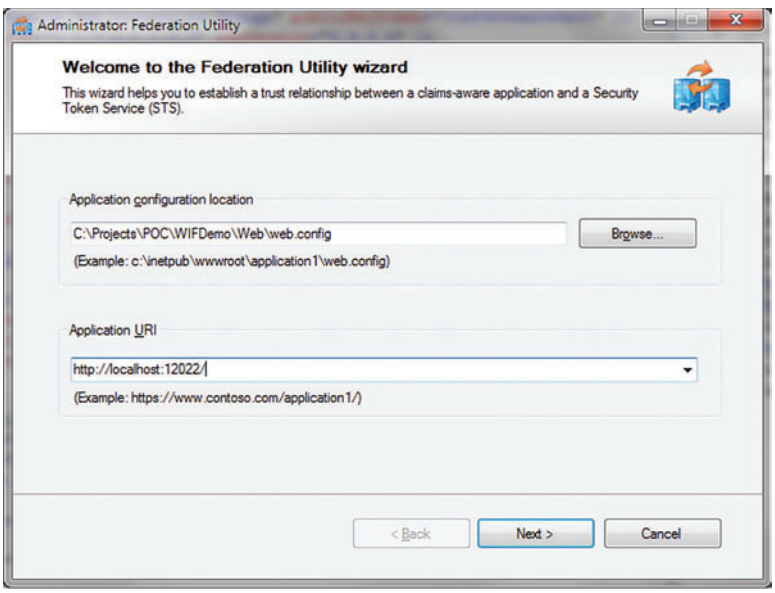


Figure 3 **Starting the Federation Utility Wizard**

Imagine...

...an intranet employees want to use

Why is user adoption such a large hurdle for intranets?

eIntranet overcomes this hurdle by transforming the user experience. Employees connect with the right people and content instantly. Information finds them, no matter where they go.

- **Collaboration** – Complete projects faster in collaborative groupspaces with powerful communication and sharing tools
- **Timeline and Social Navigation** – Find content and collateral based on when it was created and who is using it
- **Easy to deploy, customize and extend** – Integrate with business infrastructures and extend the functionality to meet unique needs
- **Mobile engagement** – Engage employees on the go, delivering updates via SMS alerts, e-mail or the eIntranet Mobile App

Learn more:

eIntranet™
Built on Ektron

http://www.ektron.com/intranet

ektron
What do you want your website to do?

The primary challenge the company faced was the length of time it took to provision a new Web server for its environment: months! When the company considered migrating the application to Windows Azure, its primary motivation was saving a tremendous amount of time. Scaling out would become a configuration detail rather than a quarter-long nightmare.

The migration process to Windows Azure is actually quite straightforward. Here's the generic process we used:

1. Verify that the application is running correctly in the development environment.
2. Migrate the SQL Server back end to SQL Azure using the SQL Azure Migration Wizard (we'll discuss the details later in this article).
3. Update the local application to work with the SQL Azure database. This was as simple as changing the connection string.
4. Convert the application into a Web Role project.
5. Validate that the application runs on the local development fabric.
6. Package up the Web Role and deploy it to Windows Azure.
7. Validate that the application runs from Windows Azure.

In order to reduce the size of the Web role package, we ended up pulling all the images and CSS files out of their content folders and placing them in Windows Azure blob storage.

Because all the content was in Windows Azure blob storage, GGP was able to leverage the Windows Azure content-delivery network (CDN). This allowed caches of data to sit closer to the end users.

For an overview of development, testing and deployment for Windows Azure, see the article "Windows Azure: Developing and Deploying Windows Azure Apps in Visual Studio 2010" in the April 2010 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee336122). For a deeper look at storage issues, see "Cloud Storage: Fueling Your Application's Engine with Windows Azure Storage" in the January 2010 issue (msdn.microsoft.com/magazine/ee335721).

Database Compatibility

When SQL Azure first came out, we migrated a couple of our SQL Server databases to it. Along with our experience holding the Windows Azure Migration Labs, we learned a few important things you should consider before embarking on the migration process.

First, it's important to check the size of your database and how it fits within the database allowances used by SQL Azure. Currently, SQL Azure offers Web Editions in 1GB and 5GB sizes and Business Editions in 10, 20, 30, 40 and 50GB sizes. You need to check your database and make sure it isn't larger than 50GB. If your database is larger than 50GB, then you'll need to examine your database and see if it can be broken down into smaller databases (in other words, sharding your database) or moving large data to blobs.

SQL Azure supports only SQL Authentication, so you'll need to consider whether changes are needed to the authentication scheme used by your application. On top of that, SQL Azure has a resource throttle that limits connection time. We'll discuss both of these issues a little later in the article.

The version of your SQL Server database is another item you need to take into consideration before migrating your database to SQL Azure. SQL Azure is built on top of SQL Server 2008. This means that if you want to migrate your SQL Server 2000 or SQL Server 2005 databases to SQL Azure, you need to make sure your databases are compatible with SQL Server 2008. For example, earlier versions of SQL Server support the old style TSQL joins such as *= and =* operators in the WHERE clause. SQL Server 2008 only supports ANSI style joins. For example:

```
SELECT ProcessClassTypeName
      , bpa.PropertyMetadata AS PropertyMetadataOverride
      , act.PropertyMetadata AS PropertyMetadataDefault
  FROM dbo.BusinessProcessActivities bpa
  LEFT JOIN dbo.Activities act ON act.Activity_ID = bpa.Activity_ID
```

When the compatibility level of the database is set to SQL Server 2005 or SQL Server 2008, the old style TSQL joins (*= and =*) are not supported. This is only one example of compatibility issues you'll find when migrating to SQL Server 2008.

It's beyond the scope of this article to discuss in detail the migration process to SQL Server 2008. If you're interested in database migration best practices, please check out the "Ultimate guide for upgrading to SQL Server 2008" (microsoft.com/downloads/details.aspx?FamilyID=66d3e6f5-6902-4fdd-af75-9975aea5bea7). There's also a wealth of resources available from the MSDN SQL Server developer center (msdn.microsoft.com/sqlserver).

You'll find the best path is to migrate from a SQL Server 2008-compatible database to SQL Azure. This means that if you want to migrate your SQL Server 2000 or 2005 database to SQL Azure, you can go through an on-premises upgrade to SQL Server 2008 before you migrate to SQL Azure.

Microsoft offers a great tool called SQL Server Upgrade Advisor (microsoft.com/downloads/details.aspx?FamilyId=F5A6C5E9-4CD9-4E42-A21C-7291E7F0F852) that analyzes instances of SQL Server 2000 and SQL Server 2005 to identify features and configuration changes that might affect your upgrade. It provides links to documentation that describes each identified issue and how to resolve it. Once you've verified that your database is compatible with SQL Server 2008, you can fast-forward to migrating the database to SQL Azure.

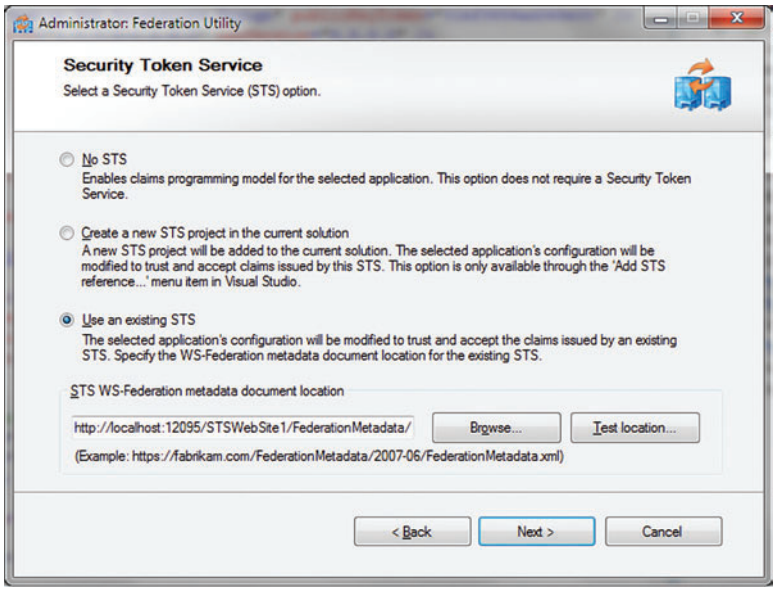


Figure 4 **Configuring the STS**

GET THE FASTEST CONTROLS...

At Infragistics, we make sure our **NetAdvantage for .NET** controls make every part of your User Interface the very best it can be. That's why we've tested and re-tested to make sure our **Data Grids are the very fastest** grids on the market and **our Data Charts outperform** any you've ever experienced. Use our controls and not only will you get the fastest load times, but your apps will always look good too. Fast and good-looking...that's a killer app. Try them for yourself at **infragistics.com/wow**.

Infragistics®
KILLER APPS. NO EXCUSES.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111
twitter.com/infragistics

That said, you also need to be aware that SQL Azure doesn't support 100 percent of new SQL Server 2008 functionality. For example, filestream is not currently supported in SQL Azure. There are a few ways to check for compatibility issues when going to SQL Azure.

The blunt force approach is to simply ride the wild side—run your TSQL scripts against SQL Azure and look for errors. Correct any errors that occur and run again. Repeat until successful. Perhaps not the best use of your time, but that's for you to decide.

You can use the SQL Server Management Studio script generator wizard to generate your TSQL script. Note that when you walk through the wizard, be sure to select the advanced scripting option and select SQL Azure Database for the "Script for the database engine type" property. If you miss this step, SQL Server will generate TSQL that's not compatible with SQL Azure.

Another option is to download SQL Azure Migration Wizard (SQLAzureMW) from sqlazuremw.codeplex.com. SQLAzureMW will do its best to identify compatibility issues, fix them where possible and notify you of all issues it knows about.

To better understand general guidelines and limitations to SQL Azure, see msdn.microsoft.com/library/ee336245.

Once you have your database schema (tables, views, stored procedures and so on) in SQL Azure, you'll need to upload your data. Here are the most common ways:

- SQL Server Integration Services
- Bulk Copy Program (BCP)
- SqlBulkCopy for data migration
- SQL Azure Migration Wizard (which uses BCP in the background)

Using SQLAzureMW

George created SQLAzureMW to help our customers with the SQL database migration process. **Figure 6** shows SQLAzureMW in action.

SQLAzureMW analyzes SQL Server databases for compatibility issues with SQL Azure. It also allows you to migrate database objects and data from the source database to SQL Azure.

By using SQLAzureMW, database developers can get an idea of how much work will be required in migrating their databases to SQL Azure. If SQLAzureMW flags a lot of compatibility issues with a SQL Server 2000 or 2005 database, we recommend upgrading your database to SQL Server 2008 first, then migrating to SQL Azure. The process of migrating to SQL Server 2008 is well-documented and there's a lot of guidance and expertise you can leverage. For more information on migrating to SQL Server 2008, see the SQL Server 2008 Upgrade Technical Reference Guide (microsoft.com/downloads/details.aspx?FamilyID=66d3e6f5-6902-4fdd-af75-9975aea5bea7). There's also a wealth of resources available from the MSDN SQL Server developer center (msdn.microsoft.com/sqlserver).

Note that if you don't have SQL Server 2008 R2, this doesn't stop you from going through the upgrade process. Just download SQL Server 2008 R2 Express Edition and go through the side-by-side upgrade process.

Other good sources for database developers to understand the differences between SQL Server and SQL Azure—what's compatible and what's not, and general guidance and limitations—include the Transact-SQL Reference (SQL Azure Database) at msdn.microsoft.com/library/ee336281 and the General Guidelines and Limitations (SQL Azure Database) at msdn.microsoft.com/library/ee336245.

No matter whether you decide to upgrade to SQL Server 2008 first or just migrate directly from SQL Server 2000 or 2005, you'll still need a way to analyze your database for compatibility issues and generate SQL Azure-compatible SQL. This is where SQLAzureMW really helps out. Not only can SQLAzureMW analyze the database, it will also analyze SQL Profiler trace files when you want to check dynamic SQL for compatibility issues.

During the migration labs, we were able to migrate all of the SQL databases (SQL Server 2000 and SQL Server 2005) to SQL Azure with little to no modification. The two remaining issues that needed to be addressed were authentication and SQL Azure resource throttling.

The authentication issue was due to the fact that SQL Azure supports only SQL Authentication, not Windows Authentication. In the case of one customer, they had to modify their connection string to reflect username and password instead of a trusted connection. For example, we started out with something like this:

```
<add key="ConStr"
  value="server=DbSvr;database=CRMDB;Trusted_Connection=yes" />
We simply changed the connection string to something like this:
<add key="ConStr"
  value="Server=avl6qnn22s.database.windows.net;Database=CRMDB;User
ID=WebSvrAdmin@avl6qnn22s;Password=password;Trusted_Connection=False;" />
To get more information on connecting to SQL Azure using ADO.NET, see
msdn.microsoft.com/library/ee336243.
```

Resource Throttling

Addressing SQL Azure resource throttling took a little more work for some applications. For the applications that followed best practices in getting a connection to a SQL database only when needed and at the last possible second, doing all of the transactions in a quick and efficient manner and letting go of the connection as soon as possible, SQL Azure throttling wasn't an issue. On the other hand, the applications that grabbed a connection to a SQL database at startup and held onto the connection for the life of the program, or for long periods of time, had to be modified to implement retry logic or refactored to follow best practices and not hold resources.

One misconception we ran into a lot was that SQL Azure would only disconnect your connection if the connection sat idle for five minutes. SQL Azure takes into account several factors in determining when to disconnect an application, including excessive resource usage, long-running queries, long-running single transactions and idle connections.

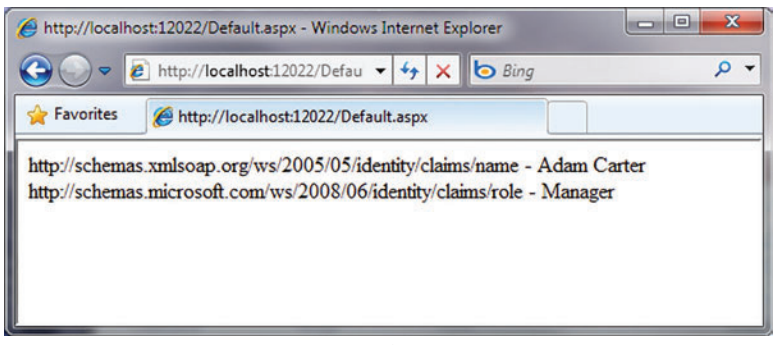
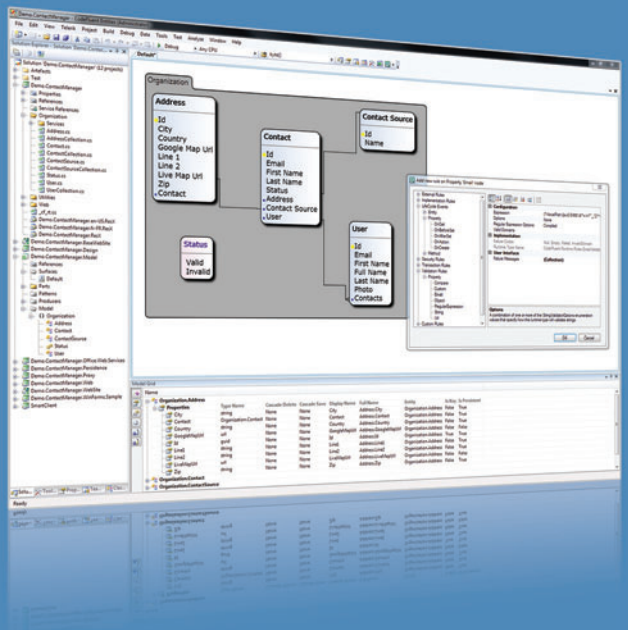


Figure 5 **The Claims Type and Value of an Identity Object**

Need a HAND?
Use CODEFLUENT ENTITIES!



APPLICATION BLOCKS

- *Localization*
- *Data Binding*
- *Rules and Validation*
- *Concurrency*
- *Security*
- *Caching*
- *Blob handling*

SUPPORTED ARCHITECTURES

- *SOA, SmartClient*
- *Rich Client, RIA, Silverlight,*
- *Web, Webparts*
- *Client/Server, N-Tier*
- *Office*
- *SharePoint*
- *SaaS, Cloud*

FEATURED TECHNOLOGIES

- *.NET (2 to 4), C#, Linq*
- *ASP.NET (WebForms, MVC)*
- *Silverlight (2 to 4)*
- *WPF, WinForms*
- *WCF, ASMX*
- *SQL Server (2000 to 2008)*
- *Oracle Database (9 to 11)*
- *Office (97 to 2010)*
- *SharePoint (2007 to 2010)*
- *Visual Studio (2005 to 2010)*

CUSTOMER APPROVED MODEL-DRIVEN SOFTWARE FACTORY

We understand the challenges that come with today's and tomorrow's technology integration and evolution.

CodeFluent Entities is a fully integrated Model-Driven Software Factory which provides architects and developers a structured method and the corresponding tools to develop .NET applications, based on any type of architecture, from an ever changing business model and rules, at an unprecedented productivity level.

CodeFluent Entities is based on a pluggable producer logic, which, from a declarative model you designed, continuously generates ready-to-use, state-of-the-art, scalable, high-performance, and easily debuggable source code, components, and everything you need.

*Download your **FREE** trial today at:*
www.CodeFluentEntities.com/Msdn

CodeFluent Entities
the Model Driven Software factOry

Contact: info@softfluent.com
www.CodeFluentEntities.com

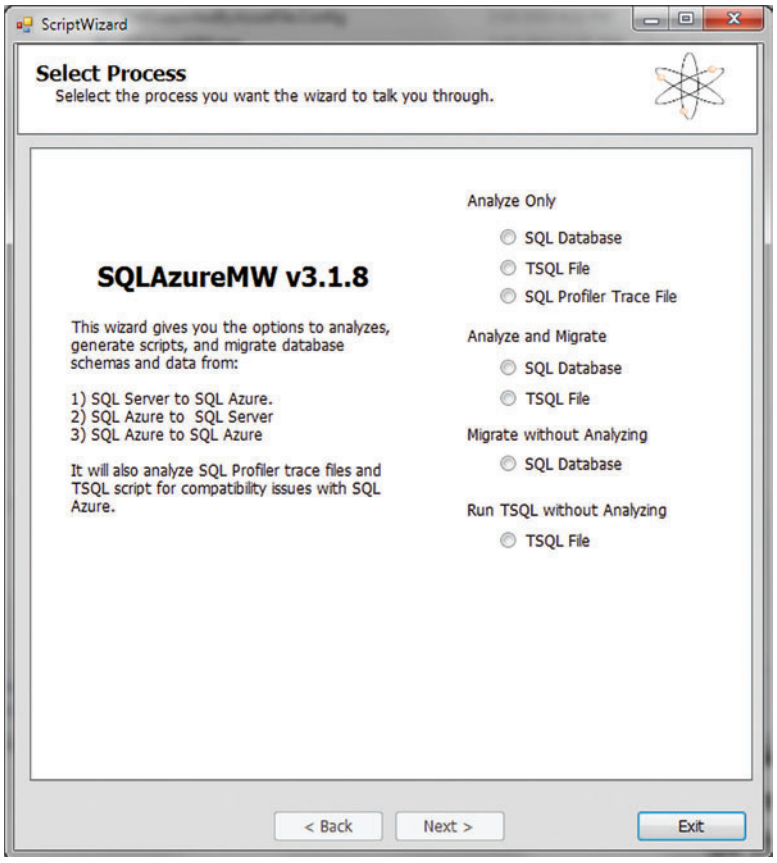


Figure 6 **Using SQLAzureMW**

The SQL Azure team will continue to tweak the resource throttling parameters, but the net effect is that the application has to have retry logic built into it because SQL Azure will force a disconnection on any application that exceeds its resource utilization parameters.

In general, SQL Azure will provide specific error messages if it ever throttles the connection. For a full list of errors, see msdn.microsoft.com/library/ff394106.

If you have a high volume of small transactions, you should use the following pattern:

1. Use the connection pool. The connection pool manager will keep the connections open for you and there will be little to no impact on performance for the application to open and close connections.
2. Keep the connection for as small a duration as possible. Open the connection, execute your transaction and close the connection.
3. Use the try-catch pattern around your database activity.
4. Catch the exceptions and retry the transaction, if appropriate.
5. Log your failures and exceptions to help facilitate problem resolution. Make sure to get a UTC timestamp (or provide time and time zone), the connection context ID and the exception number.

SQLAzureMW is a good example of an application that had to deal with resource throttling in SQL Azure. As we mentioned earlier, SQLAzureMW can migrate an on-premises SQL database to SQL Azure. If you're migrating a database that has more than 1,000 tables, 1,500 stored procedures and millions of rows of data, it could easily take longer than five hours depending on the actual amount of data that needs to be uploaded.

In this scenario, SQLAzureMW would have well over 2,500 TSQL statements to execute against SQL Azure, as well as the data to migrate via BCP. Executing more than 2,500 TSQL statements in a single statement (or transaction) would more than likely exceed SQL Azure resource throttling parameters, thus resulting in a terminated connection.

As a solution, SQLAzureMW breaks down the transactions into smaller bunches and runs until SQL Azure terminates the connection. When SQLAzureMW encounters the connection error, it reestablishes a new connection with SQL Azure and picks up processing after the last successful command. In the same manner, when using BCP to upload the data to SQL Azure, SQLAzureMW chunks the data into smaller sections and uses retry logic to figure out the last successful record uploaded before the connection was closed. Then it has BCP restart the data upload with the next set of records.

The SQL Azure product team has made vast improvements to SQL Azure since SQL Azure was first released. For example, a lot of the throttling issues we ran into during the migration labs have gone away—though we still recommend that your applications use retry logic to handle terminated connections gracefully.

Next Steps

As you can see, although there are a number of issues you need to consider in planning for a smooth Windows Azure migration, in practice we've found that the amount of work required to migrate an application from on-premises to Windows Azure is often minimal. Of course, this is going to be different for each application.

You'll need to do your own analysis to determine whether it makes sense to migrate to Windows Azure and what issues you'll need to address. During the Windows Azure Migration Labs, our customers found they were able to migrate their applications with little to no modification and that they were able to utilize the Windows Azure platform with very little learning curve and investment. The information here, along with tools like SQLAzureMW, should help you achieve a similarly successful result. ■

GEORGE HUEY *is a principal architect for the Developer & Platform Evangelism Group at Microsoft. Huey works with companies to help them understand new and emerging technologies and how these technologies can be applied to solve their business problems. He also is the author of SQL Azure Migration Wizard (SQLAzureMW).*

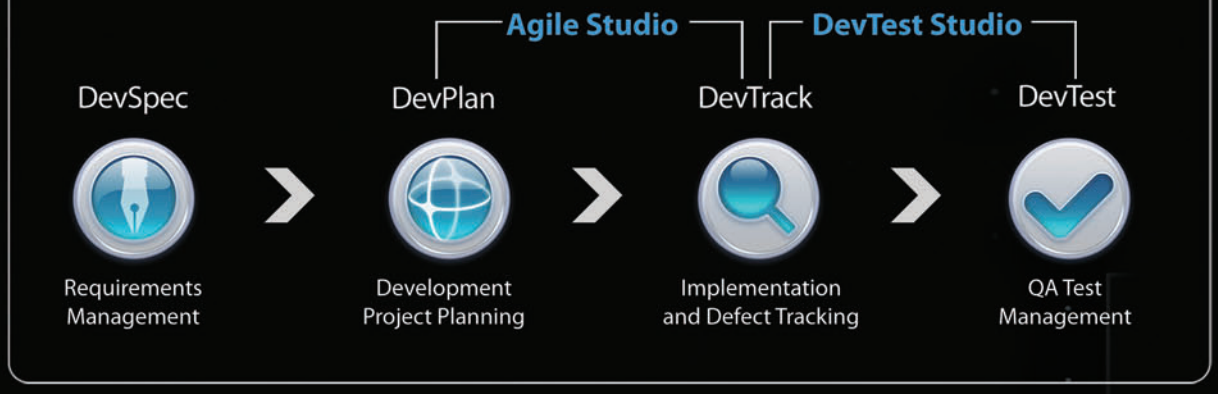
WADE WEGNER *works for Microsoft and is the technical evangelist for the Windows Azure platform. You can reach him through his blog at blog.wadewegner.com or on Twitter at twitter.com/wadewegner.*

THANKS to the following technical expert for reviewing this article: *Jim Nakashimi*

Gain Full Traceability

Requirements Driven Quality Management

DevSuite



A Modular Approach to ALM

- DevSuite can be purchased and deployed as a complete ALM solution or as individual modules
- Convenient bundles provide solutions for quality management (DevTest Studio) and Agile development (Agile Studio)
- Individual modules and bundles can be easily expanded when you need additional functionality

Agile Ready

- DevSuite is a hybrid platform that allows you to mix elements from multiple methods, including traditional development, to achieve the right balance for your team
- Out-of-the box methodology templates for Scrum, XP, Test Driven, Waterfall, and Iterative development

1-800-439-7782

www.techexcel.com

Creating Synchronization Providers with Sync Framework

Joydip Kanjilal

Microsoft Sync Framework is a comprehensive platform for synchronizing offline and online data, and facilitates collaboration and offline access for applications, services and devices alike. It is protocol- and database-independent and includes technologies and tools that enable device roaming, sharing and the ability to take networked data offline before synchronizing it back at a later point in time.

Sync Framework can be used to build applications that synchronize data from any data store using any protocol over a network. It's a comprehensive synchronization platform that facilitates offline and online data access for applications, services and devices. Sync Framework has an extensible provider model and can be used with both managed and unmanaged code to synchronize data between two data sources.

This article discusses:

- Sync Framework basics
- Default synchronization providers
- Filtering files, handling conflicts
- Creating a custom provider

Technologies discussed:

Visual Studio 2010, Sync Framework

Code download available at:

code.msdn.microsoft.com/mag201008Sync

This article takes a look at the concepts of synchronization and how Sync Framework can be integrated into your own projects. Specifically, I'll be discussing the fundamentals of data synchronization, the architectural components of Sync Framework and how you use sync providers.

To work with Sync Framework and the code examples in this article, you'll need to install Visual Studio 2010 and the Sync Framework runtime 2.0 or later. You can download the runtime with the Microsoft Sync Framework 2.0 Redistributable Package from the Sync Framework Developer Center (msdn.microsoft.com/sync).

Sync Framework Basics

Sync Framework comprises four primary components: a runtime, metadata services, synchronization providers and participants.

The Sync Framework runtime provides the infrastructure for synchronizing data between data sources. It also provides an SDK that developers can extend to implement custom providers.

Metadata services provide the infrastructure to store sync metadata, which contains information used during a synchronization session. Sync metadata includes versions, anchors and change detection information. You'll also use sync metadata in the design and development of custom providers.

Synchronization providers are used to synchronize data between replicas or endpoints. A replica is a unit of synchronization and is used to denote the actual data store. As an example, if you're synchronizing data between two databases, then each of the databases

is referred to as a replica. A replica is identified using a unique identifier called a replica key. An endpoint here also refers to a data store. I'll discuss providers in more depth later in the article.

A participant refers to the location where the data to be synchronized can be retrieved. These can be full participants, partial participants and simple participants.

Full participants are devices that can create new data stores, store sync metadata information and run sync applications on the devices themselves. Examples of full participants include desktop computers, laptops and tablets. A full participant can synchronize data with another participant.

Partial participants are devices that can create new data stores and store sync metadata information, but cannot run applications on their own. A USB storage device or smartphone could be a partial participant. Note that a partial participant can synchronize data with a full participant, but not with another partial participant.

Simple participants include devices that cannot store new data or execute applications, but can only provide the requested information. Examples of simple participants include RSS feeds and Amazon and Google Web services.

Synchronization Providers

A synchronization provider is a component that can participate in a synchronization process and enables a replica to sync data with other replicas. You should have one synchronization provider per replica.

To synchronize data, a synchronization session is started. The application connects the source and destination synchronization providers in the session to facilitate data synchronization between the replicas.

When a synchronization session is in progress, the destination provider provides information about its data store to the source provider. The source provider determines what changes to the source replica are not known to the destination replica, and then pushes the list of such changes to the destination provider. The destination provider then detects any conflicts between its own items and those present in the list, and then applies the changes to its data store. The Sync Framework engine facilitates all of this synchronization process.

Sync Framework provides support for three default providers for database, file system and feed synchronization:

- Synchronization provider for ADO.NET-enabled data sources
- Synchronization provider for RSS and Atom feeds
- Synchronization provider for files and folders

You can also extend Sync Framework to create your own custom sync provider to exchange information between devices and applications.

The database synchronization provider (previously called Sync Services for ADO.NET in Sync Framework 1.0) supports synchronization of ADO.NET-enabled data sources. You can build disconnected data applications that facilitate synchronization between ADO.NET-enabled data sources such as SQL Server. It enables roaming, sharing and taking data offline. Any database that makes use of the database provider can participate in the synchronization process with other data sources that are supported by Sync Framework including file systems, Web services or even custom data stores.

The Web synchronization provider (formerly Sync Services for FeedSync) supports synchronization of RSS and ATOM feeds. Before FeedSync, this technology was known as Simple Sharing Extensions and was originally designed by Ray Ozzie. Note that the Web synchronization provider doesn't replace the existing technologies like RSS or Atom feeds. Rather, it provides you a simple way to add synchronization capabilities to existing RSS or Atom Feeds so that they can be consumed by other applications or services independent of the platform or device in use.

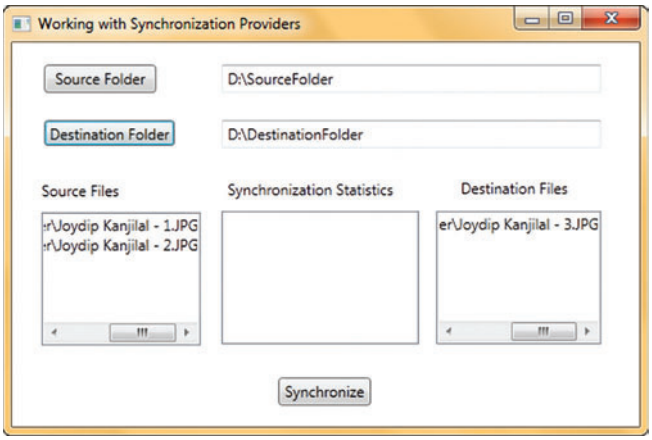


Figure 1 **The Sample Sync App**

The file synchronization provider (formerly Sync Services for File Systems) supports synchronization of files and folders in your system. It can be used to synchronize files and folders in the same system or across systems in the network. You can synchronize files and folders in systems with NTFS, FAT or SMB file systems. The provider uses the Sync Framework metadata model to enable peer-to-peer synchronization of file data with support for arbitrary topologies (client/server, full mesh and peer-to-peer) including support for removable media. The file synchronization provider also enables incremental synchronization, conflict and change detection, synchronization in both preview and non-preview modes of operation, and filtering and skipping files in the synchronization process.

Working with Built-In Sync Providers

In this section I'll demonstrate how to work with the built-in synchronization providers to implement a simple application that synchronizes the content of two folders in your system.

The FileSyncProvider class can be used to create a file synchronization provider. This class extends the UnManagedSyncProvider class and implements the IDisposable interface. The FileSyncScopeFilter class is used to include or exclude files and folders that will be participating in the synchronization process.

FileSyncProvider detects the changes in replica using sync metadata. Sync metadata contains information about all the files and folders that participate in the synchronization process. There are actually two kinds of sync metadata: replica metadata and item metadata. The file synchronization provider stores the metadata for all files and folders that participate in the synchronization process. Later, it uses the file size, attributes and the last accessed times of these files and folders to detect changes.

Open Visual Studio 2010 and create a new Windows Presentation Foundation (WPF) project. Save the project with the name SyncFiles. Open the MainWindow.xaml file and create a WPF form similar to what is shown in **Figure 1**.

Figure 2 **GetReplicaID**

```
private Guid GetReplicaID(string guidPath) {
  if (!File.Exists(guidPath)) {
    Guid replicaID = Guid.NewGuid();
    using (FileStream fileStream =
      File.Open(guidPath, FileMode.Create)) {
      using (StreamWriter streamWriter =
        new StreamWriter(fileStream)) {

        streamWriter.WriteLine(replicaID.ToString());
      }
    }

    return replicaID;
  }
  else {
    using (FileStream fileStream =
      File.Open(guidPath, FileMode.Open)) {
      using (StreamReader streamReader =
        new StreamReader(fileStream)) {

        return new Guid(streamReader.ReadLine());
      }
    }
  }
}
```

As you can see, you have controls to pick the source and destination folders. You also have controls to display the synchronization statistics and content of the source and the destination folders.

Right-click on the project in Solution Explorer, click Add Reference and add the Microsoft.Synchronization assemblies.

Now add a new GetReplicaID method in MainWindow.xaml.cs file to return a GUID as shown in the code in **Figure 2**. The Synchronize method, when called on the instance of SyncOrchestrator, creates a metadata file called filesync.metadata in each of the folders or replicas using the unique GUID. The GetReplicaID method persists this GUID in a file so that the next call to this method doesn't generate a new GUID for that particular folder. The GetReplicaID method first checks whether the file containing a replica ID exists. If the file isn't found, a new replica ID is created and stored in the file. If the file exists (because a replica ID for that folder was previously generated), it returns the replica ID from the file.

Next, add a method called GetFilesAndDirectories to return a list of the files and folders under the replica location (see **Figure 3**). The folder name should be passed to it as a parameter.

> Synchronization providers are used to synchronize data between replicas or endpoints.

This method would be used to display the list of files and folders inside the source and destination folders both before and after the synchronization process. The methods PopulateSourceFileList and PopulateDestinationFileList call GetFilesAndDirectories to populate the list boxes that display the files and directories inside the source and destination folders (see the code download for details).

The btnSource_Click and the btnDestination_Click event handlers are used to select the source and the destination folders. Both methods make use of the FolderBrowser class to display a dialog box from where the user can select the source or destination folders. The complete source code of the FolderBrowser class is available for download with the code download for this article.

Now I need to write the Click event handler of the Button control, which starts by disabling the button before synchronization starts. It then calls the Synchronize method with the source and destination paths as parameters. Finally, I start the synchronization process, catch any errors, and enable the button when synchronization completes:

```
btnSyncFiles.IsEnabled = false;
// Disable the button before synchronization starts
Synchronize(sourcePath, destinationPath);
btnSyncFiles.IsEnabled = true;
// Enable the button after synchronization is complete
```

The Synchronize method accepts the source and destination path and synchronizes content of the two replicas. In the Synchronize method, I take an instance of the SyncOperationStatistics class to retrieve statistical information on the synchronization process:

```
SyncOperationStatistics syncOperationStatistics;
```

I also create the source and destination sync providers, create a SyncOrchestrator instance named synchronizationAgent, assign the GUIDs to the source and destination replicas and attach the two providers to it. The SyncOrchestrator is responsible for coordinating the synchronization session:

```
sourceReplicaID =
  GetReplicaID(Path.Combine(source,"ReplicaID"));
destinationReplicaID =
  GetReplicaID(Path.Combine(destination,"ReplicaID"));

sourceProvider =
  new FileSyncProvider(sourceReplicaID, source);
destinationProvider =
  new FileSyncProvider(destinationReplicaID, destination);

SyncOrchestrator synchronizationAgent =
  new SyncOrchestrator();
synchronizationAgent.LocalProvider = sourceProvider;
synchronizationAgent.RemoteProvider = destinationProvider;
```

Finally, I start the synchronization process, catch any errors and release resources as appropriate as shown in **Figure 4**. The code download for this article includes the complete source project with error handling and other implementation details.

You can also report the synchronization progress for a synchronization session. To implement this, follow these steps:

1. Register an event handler for the ApplyingChange event.
2. Enable preview mode by setting the PreviewMode property of FileSyncProvider to true.
3. Take an integer counter and increase it each time the ApplyingChange event is triggered.
4. Start the synchronization process.
5. Set the PreviewMode property of FileSyncProvider to false to disable PreviewMode.
6. Start the synchronization process again.

Filtering and Skipping Files

When synchronizing using Sync Framework, some files are skipped automatically, including Desktop.ini and Thumbs.db, files with system and hidden attributes, and metadata files. You can apply static filters to control the files and folders you want to be synchronized. Specifically, these filters exclude the files you don't want to be a part of the synchronization process.

Does your Team do more than just track bugs?

Free Trial and Single User FreePack™ available at www.alexcorp.com

Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

Track all your project tasks in one database so you can work together to get projects done.

- Quality Control / Compliance Tracking
- Project Management
- End User Accessible Service Desk Portal
- Bugs and Features
- Action Items
- Sales and Marketing
- Help Desk

Native Smart Card Login Support including Government and DOD

New in Team 2.11

- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

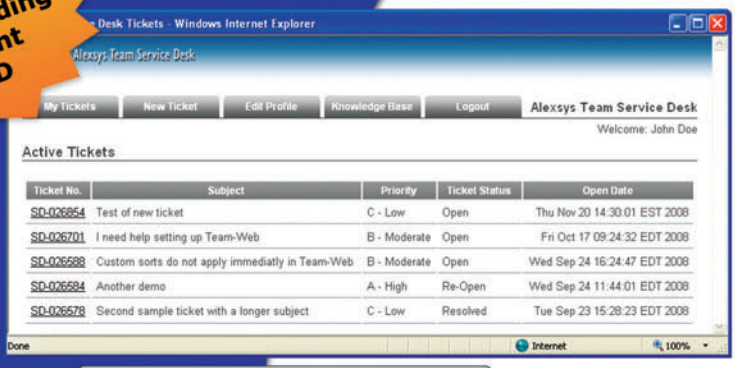
Service Desk Features

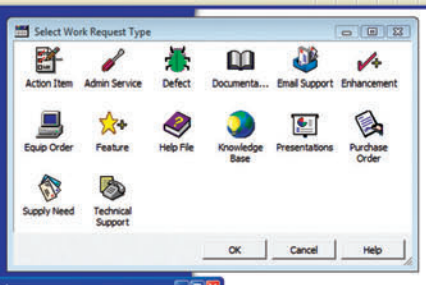
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

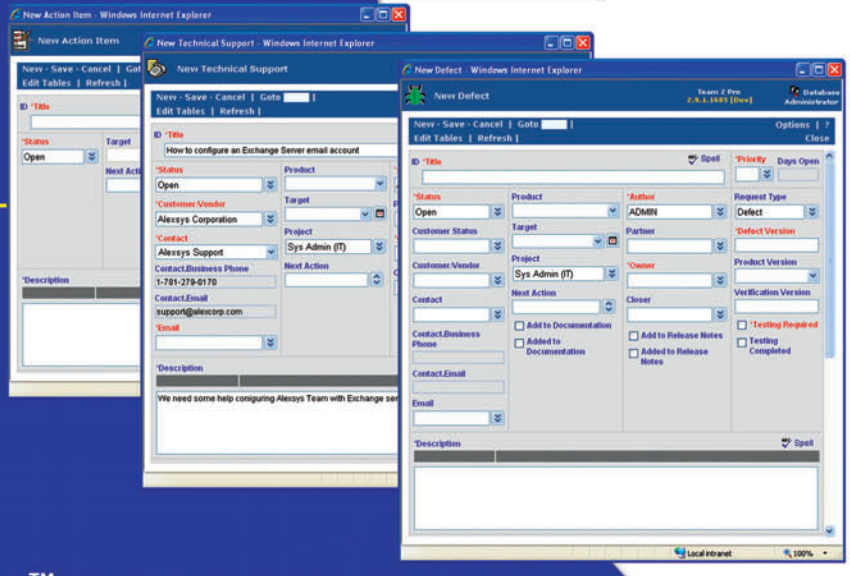
Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick







Free Trial and Single User FreePack™ available at www.alexcorp.com.
FreePack™ includes a free single user Team Pro and Team-Web license.
Need more help? Give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team Pro works with Microsoft SQL, MySQL, and Oracle Servers.
Team 2 works with Windows 7/2008/2003/Vista/XP.
Team-Web works with Internet Explorer, Firefox, Netscape, Safari, and Chrome.

To use static filters, create an instance of the FileSyncScopeFilter class and pass the inclusion and exclusion filters as parameters to its constructor. You can also use the FileNameExcludes.Add method on your FileSyncScopeFilter instance to filter out one or more files from the synchronization session. You can then pass in this FileSyncScopeFilter instance when creating your FileSyncProvider instance. Here's an example:

```
FileSyncScopeFilter fileSyncScopeFilter =
  new FileSyncScopeFilter();
fileSyncScopeFilter.FileNameExcludes.Add("filesync.id");
FileSyncProvider fileSyncProvider =
  new FileSyncProvider(Guid.NewGuid(),
  "D:\\MyFolder",fileSyncScopeFilter,FileSyncOptions.None);
```

Similarly, you can exclude all .lnk files from the synchronization process:

```
FileSyncScopeFilter fileSyncScopeFilter =
  new FileSyncScopeFilter();
fileSyncScopeFilter.FileNameExcludes.Add("*.lnk");
```

You can even use FileSyncOptions to explicitly set options for the synchronization session:

```
FileSyncOptions fileSyncOptions =
  FileSyncOptions.ExplicitDetectChanges |
  FileSyncOptions.RecycleDeletedFiles |
  FileSyncOptions.RecyclePreviousFileOnUpdates |
  FileSyncOptions.RecycleConflictLoserFiles;
```

To skip one or more files during the synchronization process, register an event handler on the ApplyingChange event and set the SkipChange property to true:

```
FileSyncProvider fileSyncProvider;
fileSyncProvider.AppliedChange +=
  new EventHandler (OnAppliedChange);
destinationProvider.SkippedChange +=
  new EventHandler (OnSkippedChange);
```

Now I can implement the OnAppliedChange event handler to show what changes occur:

```
public static void OnAppliedChange(
  object sender, AppliedChangeEventArgs args) {
  switch (args.ChangeType) {
    case ChangeType.Create:
      Console.WriteLine("Create " + args.NewFilePath);
      break;
    case ChangeType.Delete:
      Console.WriteLine("Delete" + args.OldFilePath);
      break;
    case ChangeType.Overwrite:
      Console.WriteLine("Overwrite" + args.OldFilePath);
      break;
    default:
      break;
  }
}
```

Note that this example is simplified for clarity. A more robust implementation is included in the code download.

To understand why a particular file has been skipped during the synchronization session, you can implement the OnSkippedChange event handler:

```
public static void OnSkippedChange(
  object sender, SkippedChangeEventArgs args) {

  if (args.Exception != null)
    Console.WriteLine("Synchronization Error: " +
      args.Exception.Message);
}
```

Build and execute the application. Click on the Source Folder button to select the source folder. Use the Destination Folder to select the destination folder. You'll see the list of the files in each of the folders before synchronization is displayed in the respective list boxes (see **Figure 1**). The Synchronization Statistics list box doesn't display anything as synchronization is yet to be started.

Figure 3 **Getting Replica Files and Folders**

```
private List<string> GetFilesAndDirectories(String directory) {
  List<String> result = new List<String>();
  Stack<String> stack = new Stack<String>();
  stack.Push(directory);

  while (stack.Count > 0) {
    String temp = stack.Pop();

    try {
      result.AddRange(Directory.GetFiles(temp, "*.*"));

      foreach (string directoryName in
        Directory.GetDirectories(temp)) {
        stack.Push(directoryName);
      }
    }
    catch {
      throw new Exception("Error retrieving file or directory.");
    }
  }

  return result;
}
```

Now click the Synchronize button to start the synchronization process. Once the source and destination folders have been synchronized, you'll see the content of both folders after synchronization in the respective list boxes. The Synchronization Statistics list box now displays information about the tasks that were completed (see **Figure 5**).

Handling Conflicts

Sync Framework manages all the complexities involved in timestamp-based synchronization that include deferred conflicts, failures, interruptions and loops. To handle data conflicts when a synchronization session is in progress, Sync Framework follows one of the following strategies:

- Source Wins: In this strategy, the changes that have been made in the source data store in the event of a conflict always win.

Figure 4 **Synchronizing Replicas**

```
try {
  syncOperationStatistics = synchronizationAgent.Synchronize();

  // Assign synchronization statistics to the lstStatistics control
  lstStatistics.Items.Add("Download Applied: " +
    syncOperationStatistics.DownloadChangesApplied.ToString());
  lstStatistics.Items.Add("Download Failed: " +
    syncOperationStatistics.DownloadChangesFailed.ToString());
  lstStatistics.Items.Add("Download Total: " +
    syncOperationStatistics.DownloadChangesTotal.ToString());
  lstStatistics.Items.Add("Upload Total: " +
    syncOperationStatistics.UploadChangesApplied.ToString());
  lstStatistics.Items.Add("Upload Total: " +
    syncOperationStatistics.UploadChangesFailed.ToString());
  lstStatistics.Items.Add("Upload Total: " +
    syncOperationStatistics.UploadChangesTotal.ToString());
}
catch (Microsoft.Synchronization.SyncException se) {
  MessageBox.Show(se.Message, "Sync Files - Error");
}
finally {
  // Release resources once done
  if (sourceProvider != null)
    sourceProvider.Dispose();
  if (destinationProvider != null)
    destinationProvider.Dispose();
}
```

- Destination Wins: In this strategy, the changes that have been made in the destination data store in the event of a conflict always win.
- Merge: In this strategy, the changes in the event of a conflict are merged together.
- Log conflict: This is a strategy in which the conflict is deferred or logged.

Understanding the Synchronization Flow

A SyncOrchestrator instance controls a synchronization session and the flow of data during the session. The synchronization flow is always unidirectional and you have a source provider attached to the source replica and a destination provider attached to the destination replica. The first step is to create your source and destination providers, assign unique replica IDs to them and attach the two providers to the source and destination replicas:

```
FileSyncProvider sourceProvider =
  new FileSyncProvider(sourceReplicaID, @"D:\Source");
FileSyncProvider destinationProvider =
  new FileSyncProvider(destinationReplicaID, @"D:\Destination");
```

Next, create an instance of SyncOrchestrator and attach the two providers to it. A call to the Synchronize method on the SyncOrchestrator instance creates a link between the source and the destination providers:

```
SyncOrchestrator syncAgent = new SyncOrchestrator();
syncAgent.LocalProvider = sourceProvider;
syncAgent.RemoteProvider = destProvider;
syncAgent.Synchronize();
```

From that point, a number of calls can be made by Sync Framework while a synchronization session is in progress. Let's walk through them.

> A SyncOrchestrator instance controls a synchronization session and the flow of data.

BeginSession is called on both the source and destination providers to indicate the synchronization provider is about to join a synchronization session. Note that the BeginSession method throws InvalidOperationException if the session cannot be started or the provider is not initialized properly:

```
public abstract void BeginSession(
  SyncProviderPosition position,
  SyncSessionContext syncSessionContext);
```

Sync Framework calls GetSyncBatchParameters on the instance of the destination provider. The destination provider returns its knowledge (a compact representation of versions or changes that a particular replica is aware of) and the requested batch size. This method accepts two out parameters, namely, batchSize and knowledge:

```
public abstract void GetSyncBatchParameters(
  out uint batchSize,
  out SyncKnowledge knowledge);
```

Sync Framework invokes GetChangeBatch on the source provider. This method accepts two input parameters, the batch size and the knowledge of the destination:

```
public abstract ChangeBatch GetChangeBatch(
  uint batchSize,
  SyncKnowledge destinationKnowledge,
  out object changeDataRetriever);
```

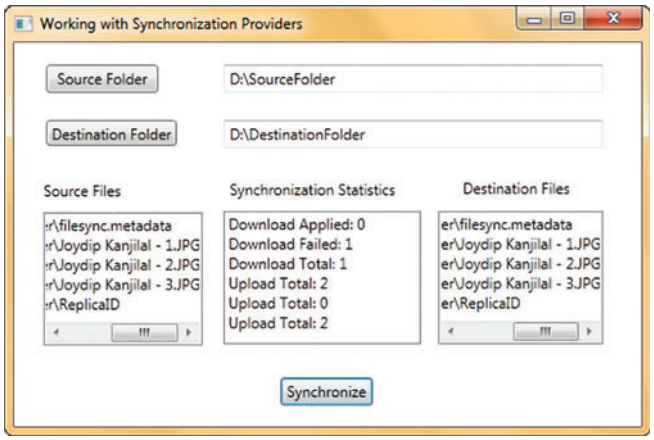


Figure 5 **Synchronization Finished**

The source synchronization provider now sends the summary of changed versions and knowledge to the destination provider in the form of changeDataRetriever object.

The ProcessChangeBatch method is called on the destination provider to process the changes:

```
public abstract void ProcessChangeBatch(
  ConflictResolutionPolicy resolutionPolicy,
  ChangeBatch sourceChanges,
  object changeDataRetriever,
  SyncCallbacks syncCallbacks,
  SyncSessionStatistics sessionStatistics);
```

SaveItemChange is called on the destination synchronization provider for each of the changes in the batch. If you're implementing your own custom provider, you should update the destination replica with the changes sent by the source replica and then update the metadata in the metadata store with the source knowledge:

```
void SaveItemChange(SaveChangeAction saveChangeAction,
  ItemChange  change, SaveChangeContext context);
```

StoreKnowledgeForScope is called on the destination synchronization provider to save knowledge in the metadata store:

```
public void StoreKnowledgeForScope(
  SyncKnowledge knowledge,
  ForgottenKnowledge forgottenKnowledge)
```

EndSession is called on both the source and destination providers to indicate that the synchronization provider is about to leave the synchronization session it joined earlier:

```
public abstract void EndSession(
  SyncSessionContext syncSessionContext);
```

Custom Synchronization Providers

Now you've seen how the default synchronization providers work. As I've mentioned before, you can also implement custom synchronization providers. A custom synchronization provider extends the functionality of a built-in synchronization provider. You may need a custom synchronization provider if there's no provider for the data stores to be synchronized. You can also create a custom synchronization provider that implements change units for better control over change tracking and to reduce the number of conflicts.

To design your own synchronization provider, create a class that extends the KnowledgeSyncProvider abstract class and implements the IChangeDataRetriever and INotifyingChangeApplierTarget interfaces. Note that these classes and interfaces are part of the Microsoft.Synchronization namespace.

As an example of a custom provider, say you wanted to implement a synchronization provider for synchronizing data between databases. This is just an overview of a simple example, and it could be extended to accommodate much more complicated scenarios.

Start by creating three databases in SQL Server 2008 (I named them ReplicaA, ReplicaB and ReplicaC) and create a table in each database called Student. The custom provider will sync records between these three Student tables. Next, create an entity called Student for performing CRUD operations on the Student table.

Create a class called Student with StudentID, FirstName, LastName as fields, and the necessary helper methods to execute CRUD operations in the database:

```
public class Student {
  public int StudentID { get; set; }
  public String FirstName { get; set; }
  public String LastName { get; set; }
  //Helper methods for CRUD operations
...
}
```

Create a class called CustomDBSyncProvider and extend it from the KnowledgeSyncProvider, IChangeDataRetriever, INotifyingChangeApplierTarget and IDisposable interfaces:

```
using System;
using System.Collections.Generic;
using System.IO;
using Microsoft.Synchronization;
using Microsoft.Synchronization.MetadataStorage;
public class CustomDBSyncProvider : KnowledgeSyncProvider,
  IChangeDataRetriever,
  INotifyingChangeApplierTarget,IDisposable {
...
```

Implement the necessary methods in your custom database synchronization provider and create the UI to display the content of each of the Student tables (see the code download for this article for details).

Now, create three instances of the custom synchronization provider and attach them to each of the Student database tables. Finally, synchronize the content of one replica with another with the help of the custom synchronization provider:

```
private void Synchronize(
  CustomDBSyncProvider sourceProvider,
  CustomDBSyncProvider destinationProvider) {

  syncAgent.Direction =
    SyncDirectionOrder.DownloadAndUpload;
  syncAgent.LocalProvider = sourceProvider;
  syncAgent.RemoteProvider = destinationProvider;
  syncStatistics = syncAgent.Synchronize();
}
```

Synced Up

As you've seen, Sync Framework provides a simple yet comprehensive synchronization platform that provides seamless synchronization between offline and online data. It can be used to synchronize data independent of the protocol and the data store in use. It could be used for simple file backup or easily extended for collaboration-based networks. You can also create custom synchronization providers to support data sources that aren't accommodated out of the box. ■

JOYDIP KANJILAL *is an independent software consultant as well as a Microsoft MVP in ASP.NET since 2007. He's also a speaker and author of several books and articles and blogs at aspadvice.com/blogs/joydip.*

THANKS to the following technical expert for reviewing this article: *Liam Cavanagh*

Building an AtomPub Server Using WCF Data Services

Chris Sells

If you're not familiar with it, the Open Data Protocol (OData) is a thing of beauty. OData (described in detail at odata.org) builds on the HTTP-based goodness of Atom for publishing data; AtomPub for creating, updating and deleting data; and the Microsoft Entity Data Model (EDM) for defining the types of data.

If you have a JavaScript client, you can get the data back directly in JSON instead of Atom format, and if you've got something else—including Excel, the .Microsoft NET Framework, PHP, AJAX and more—there are client libraries for forming OData requests and consuming OData responses. If you're using the .NET Framework on the server side, Microsoft also provides an easy-to-use library called WCF Data Services for exposing .NET Framework types or databases supported by the Microsoft Entity Framework as OData sources. This makes it easy to expose your data over the Internet in an HTTP- and standards-based way.

Having said all that, there are some things that you might like to do with OData that aren't quite part of the out-of-box experience, such as integrating OData with existing Atom- and AtomPub-based readers and writers. That's what we're going to experiment with.

This article discusses:

- OData
- Building a blogging system with ASP.NET MVC 2.0
- Mapping between Atom and OData
- Building a rich text editor with Windows Live Writer

Technologies discussed:

OData, Atom, AtomPub, ASP.NET MVC 2.0, Entity Data Model, Entity Framework, WCF Data Services

Code download available at:

code.msdn.microsoft.com/mag201008Atom

A Simple Blog

As an example, let's imagine that I'm building a simple blogging system (and, in fact, this work is based on me rewriting the content management system on sellsbrothers.com). I'm a big fan of the model-first support in Visual Studio 2010, so I created an ASP.NET MVC 2.0 project, added an ADO.NET EDM file called MyBlogDB.edmx and laid out a Post entity, as shown in **Figure 1**.

More complicated blogging software will want to track more data, but the fields in **Figure 1** are the basics. When I right-click on the designer surface, I can choose Generate Database From Model, which shows the SQL file that will be created for me (MyBlogDB.sql in this case) and the SQL that will be generated to create my database. Clicking Finish will create the SQL file and bind the database to the entities I created in the EDM designer. The important bits of the SQL are shown in **Figure 2**.

Basically, we're just creating a single table from our single entity, as expected, and mapping the fields to SQL types. Notice that the PublishDate is set to NULL, which is not the default. I explicitly chose that setting in the EDM designer because I wanted it to be

OK not to have a publish date (some tools don't provide one by default).

To execute this SQL and create the database is just a matter of right-clicking on the SQL in the Visual Studio text editor and choosing Execute SQL. It will ask you for your connection information and the database name. Because this is a new database, you'll want to type in the new name, for example, MyBlogDB, and click OK to create it when prompted. When your database has been created, you can explore it in the Server Explorer under the connection that Visual Studio has just created for you.

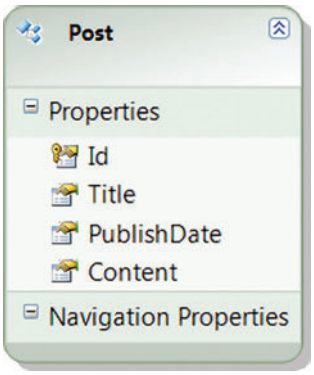


Figure 1 **A Post Entity Created in Visual Studio 2010**

To make testing easier, you can add data directly into the table by right-clicking on Posts and choosing Show Table Data, which will give you a little grid, as shown in **Figure 3**.

It's not the best editing experience in the world, but it's better than writing SQL statements until we've got an end-to-end editing solution up and running (it's coming—keep reading!).

Now that we've got some data, we can do a little bit of ASP.NET coding to show it by updating HomeController.cs (read more about MVC at asp.net/mvc/):

```
...
namespace ODataBloggingSample.Controllers {
  [HandleError]
  public class HomeController : Controller {
    MyBlogDBContainer blogDB = new MyBlogDBContainer();

    public ActionResult Index() {
      return View(blogDB.Posts);
    }

    public ActionResult About() {
      return View();
    }
  }
}
```

Here all I did was create an instance of the MyBlogDBContainer class, which is the top-level ObjectContext-derived class created from our MyBlogDB.edmx file to let us access our new database. (If you're not familiar with the Entity Framework, you should be: see msdn.com/data/aa937723.) When the Index method is called on the HomeController class, someone is requesting the home page of our new Web application, which we'd like to use to show our new blog posts, so we route the Posts collection from the database to an instance of the Home/Index.aspx view, which we've modified like so:

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
    Inherits="System.Web.Mvc.ViewPage<IEnumerable<ODataBloggingSample.Post>>" %>

<asp:Content ID="indexTitle" ContentPlaceHolderID="TitleContent" runat="server">
    Home Page
</asp:Content>
<asp:Content ID="indexContent" ContentPlaceHolderID="MainContent" runat="server">
    <% foreach (var post in Model) { %>
      <h1><%= post.Title %></h1>
      <div><%= post.Content %></div>
      <p><i>Posted <%= post.PublishDate %></i></p>
    <% } %>
</asp:Content>
```

Here we changed the base class to take a collection of the Post type generated (along with the MyBlogDBContainer class) to model our Posts table. We also replaced the home page content with a foreach statement to show each post's title, content and publish date.

That's all we need. Now when we execute the project (Debug | Start Debugging), the browser starts and the blog posts are shown (just one post, unless you've put more than that into the database), as shown in **Figure 4**.

Now, I told you everything up to this point so I could tell you this: The reason OData is so fabulous is that, with a flick of my mouse and two shakes of my keyboard, I can expose a complete programmatic interface to this data that I can access from JavaScript, the .NET Framework, PHP and more. To see this magic happen, right-click on your project in the Solution Explorer, choose Add | New Item, choose WCF Data Service, pick a name (I used odata.svc) and click Add. What you'll get is a skeleton piece of code in a file (odata.svc.cs in this case) that, ignoring security just now, we'd like to make look like the following:

```
using System.Data.Services;
using System.Data.Services.Common;
using ODataBloggingSample;

namespace ODataBloggingSample {
  public class odata : DataService<MyBlogDBContainer> {
    public static void InitializeService(DataServiceConfiguration config) {
      config.SetEntitySetAccessRule("*", EntitySetRights.All);
      config.DataServiceBehavior.MaxProtocolVersion =
        DataServiceProtocolVersion.V2;
    }
  }
}
```

Notice that we've thrown in MyBlogDBContainer—our top-level database access class—as the template parameter to the DataService class, which is the core of server-side WCF Data Services (see msdn.com/data/bb931106). The DataService class allows us to easily expose our database via HTTP verb-based create, read, update and delete (CRUD) operations defined in the OData protocol. The type passed to the DataService is examined for public properties that expose collections. In our example, the Entity Framework-generated object context class contains the Posts collection that fits the bill nicely:

```
...
namespace ODataBloggingSample {
  ...
  public partial class MyBlogDBContainer : ObjectContext {
    ...
    public ObjectSet<Post> Posts {...}
   ...
  }

  ...
  public partial class Post : EntityObject {
    ...
    public global::System.Int32 Id { get { ... } set { ... } }
    public global::System.String Title { get { ... } set { ... } }
    public Nullable<global::System.DateTime> PublishDate {
      get { ... } set { ... } }
    public global::System.String Content { get { ... } set { ... } }
    ...
  }
}
```

Notice that the generated MyBlogDBContainer exposes an ObjectSet (which is just a kind of collection) called Posts that contains instances of the Post type. Furthermore, the Post type is defined to provide the mapping between the Id, Title, PublishDate and Content properties to the underlying columns on the Posts table we created earlier.

Figure 2 **The SQL Code Resulting from "Generate Database From Model"**

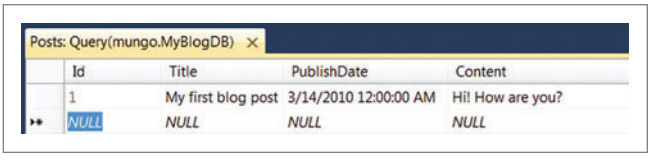
```
...
USE [MyBlogDB];
GO
...
-- Dropping existing tables
IF OBJECT_ID(N'[dbo].[Posts]', 'U') IS NOT NULL
    DROP TABLE [dbo].[Posts];
GO
...
-- Creating table 'Posts'
CREATE TABLE [dbo].[Posts] (
    [Id] int IDENTITY(1,1) NOT NULL,
    [Title] nvarchar(max)  NOT NULL,
    [PublishDate] datetime  NULL,
    [Content] nvarchar(max)  NOT NULL
);
GO
...
-- Creating primary key on [Id] in table 'Posts'
ALTER TABLE [dbo].[Posts]
ADD CONSTRAINT [PK_Posts]
    PRIMARY KEY CLUSTERED ([Id] ASC);
GO
```

With odata.svc in place, we can surf to the service document that exposes our object context collection properties using the name of the data service endpoint file in the URL, for example, localhost:54423/odata.svc:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<service xml:base="http://localhost:54423/odata.svc/"
xmlns:atom="http://www.w3.org/2005/Atom"
  xmlns:app="http://www.w3.org/2007/app" xmlns="http://www.w3.org/2007/app">
    <workspace>
     <atom:title>Default</atom:title>
      <collection>
        <atom:title>Posts</atom:title>
      </collection>
    </workspace>
</service>
```

This entire file is defined by the AtomPub specification (ietf.org/rfc/rfc5023.txt). Taking it one level deeper, we can see our posts exposed as a set of Atom entries at localhost:54423/odata.svc/Posts, as shown in **Figure 5**.

This file is almost completely plain-vanilla Atom (ietf.org/rfc/rfc4287.txt) except for the Microsoft-based URIs, which are used to layer OData functionality into Atom. Specifically, you'll want to notice the "properties" element inside the "content" element. You'll recognize these properties as the same ones defined earlier in the Post entity and corresponding Posts table. This data is contained in the envelope defined by Atom and exposed via the CRUD comments, which themselves are defined by AtomPub and allow you to create, read, update and delete via the HTTP methods POST, GET, PUT and DELETE, respectively. The problem is that this isn't quite plain-vanilla-Atom enough. For example, if we surf to odata.svc/Posts in an Atom reader, like Internet Explorer 8, the title and content don't come through properly, as shown in **Figure 6**.


Posts: Query(mungo.MyBlogDB)

Id	Title	PublishDate	Content
1	My first blog post	3/14/2010 12:00:00 AM	Hi! How are you?
NULL	NULL	NULL	NULL

Figure 3 **The Show Table Data Grid Makes Testing Easier**

You can see that the data is there (notice the date is right and the category is showing), but the title and content are nowhere to be seen. That's because the places where Internet Explorer is looking for title and content—the "title" and "content" elements in each entry, logically enough—don't contain what it expects to be there. The "title" element is blank and the "content" element is in a format that Internet Explorer doesn't recognize. The format that Internet Explorer would really like to see looks like this:

```
<feed ...>
  <title type="text">Posts</title>
  <id>http://localhost:54423/atompub.svc/Posts</id>
  <updated>2010-03-15T00:42:32Z</updated>
  <link rel="self" title="Posts" href="Posts" />
  <entry>
    <id>http://localhost:54423/atompub.svc/Posts(1)</id>
    <title type="text">My first blog post</title>
    <updated>2010-03-15T00:42:32Z</updated>
    ...
    <content type="html">Hi! How are you?</content>
    <published>2010-03-14T00:00:00-08:00</published>
  </entry>
</feed>
```

Notice that the "title" element has what used to be buried in the Title property from the OData "properties" element in the "content" element, the "content" element has been overwritten with the Content property and the "published" element was added from the value of PublishDate property. When this data is viewed in Internet Explorer, we get something much more like what we'd like to have, as shown in **Figure 7**.

I should mention that it's only for support of blogging tools that we even care. Internet Explorer isn't expecting to see a customer list or an invoice; it's expecting to see titles and publish dates and HTML content. Sometimes it makes sense to do this mapping for customer lists and invoices, in which case Microsoft has a feature in WCF Data Services called "Friendly Feeds" (see blogs.msdn.com/astoriateam/archive/2008/09/28/making-feeds-friendly.aspx). It doesn't quite do everything, however (specifically, it won't remap the Atom "content" element), because the WCF Data Services team wants to make sure even "friendly" feeds still work with various client libraries. The goal is to make the OData feeds friendly, not abandon OData in favor of Atom/AtomPub.

In this case, however, we're abandoning OData and just using WCF Data Services as our AtomPub endpoint, which requires a mapping between Atom and OData, as shown in **Figure 8**.

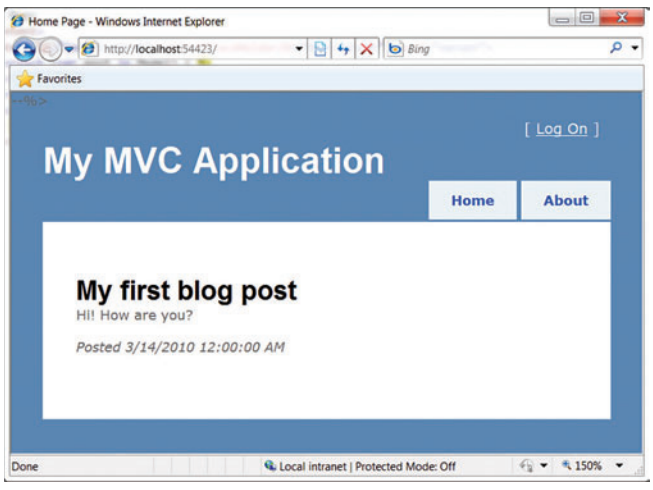


Figure 4 **The Completed Web Page**

INSPIRE! EMPOWER! IMPRESS!

You've got the data, but time, budget and staff constraints can make it hard to present that valuable information in a way that will impress. With Infragistics' **NetAdvantage for Silverlight Data Visualization**, you can create Web-based data visualizations and dashboard-driven applications on Microsoft Silverlight (and coming soon for WPF) that will not only impress decision makers, it actually empowers them. Go to infragistics.com/sldv today and get inspired to create killer apps.

Infragistics®
KILLER APPS. NO EXCUSES.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111
twitter.com/infragistics

Figure 5 **Posts Exposed as a Set of Atom Entries**

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<feed xml:base="http://localhost:54423/odata.svc/"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m=
    "http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Posts</title>
  <id>http://localhost:54423/odata.svc/Posts</id>
  <updated>2010-03-15T00:26:40Z</updated>
  <link rel="self" title="Posts" href="Posts" />
  <entry>
    <id>http://localhost:54423/odata.svc/Posts(1)</id>
    <title type="text" />
    <updated>2010-03-15T00:26:40Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Post" href="Posts(1)" />
    <category term="MyBlogDB.Post"
      scheme=
        "http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:Id m:type="Edm.Int32">1</d:Id>
        <d:Title>My first blog post</d:Title>
        <d:PublishDate m:type=
          "Edm.DateTime">2010-03-14T00:00:00</d:PublishDate>
        <d:Content>Hi! How are you?</d:Content>
      </m:properties>
    </content>
  </entry>
</feed>
```

The trick is, how do we get this mapping to happen? We've obviously got the data, but we need to remap it to Atom properties so that Atom readers (and writers) know where the data is tucked away. The reason to do this is so WCF Data Services can still do the mapping to our .NET Framework types or, via Entity Framework, our databases. All we have to do is a little at-the-door mapping of Atom/AtomPub to/from OData.

The code sample that comes with this article has some code to inject into the WCF pipeline that allows just exactly this kind of message data transformation. You can read it to your heart's content (check out ODataBlogging.cs), but I'm going to show you how to use it.

First, create a new WCF Data Services endpoint just like you did before, but with a different name (I used atompub.svc). Hook up the top-level object context class and expose whatever entity sets you like, just as before, but also tag your service class with the ODataBloggingServiceBehavior, like so:

```
...
using ODataBlogging;

namespace ODataBloggingSample {
  [ODataBloggingServiceBehavior(typeof(MyBlogDBContainer))]
  [EntityAtomMapping("Posts", "PublishDate", "published")]
  public class atompub : DataService<MyBlogDBContainer> {
    public static void InitializeService(DataServiceConfiguration config) {
      config.SetEntitySetAccessRule("*", EntitySetRights.All);
      config.DataServiceBehavior.MaxProtocolVersion =
        DataServiceProtocolVersion.V2;
    }
  }
}
```

This does the mapping from Atom/AtomPub coming in—for example, "title," "content" and "published" elements—to the corresponding OData format via the nested "properties" element inside the "content" element. By default, if the names on the entities match (ignoring case), then the mapping (and type coercion) will just happen.

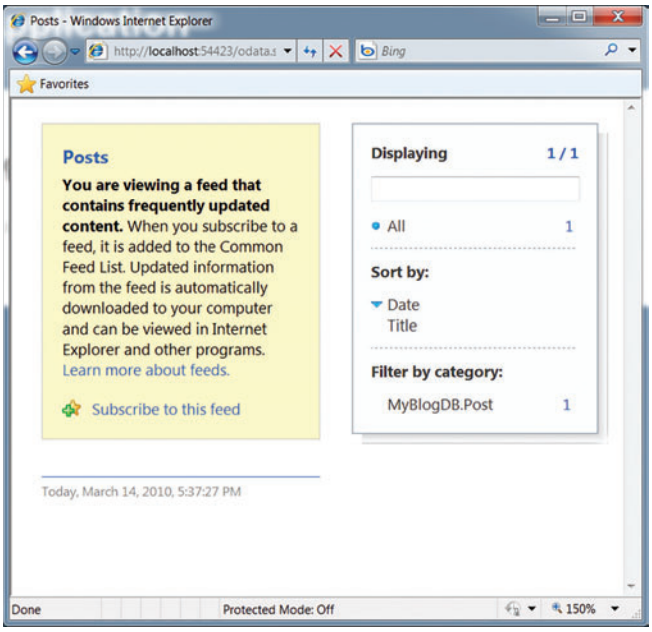


Figure 6 **Viewing Blog Posts in Atom Reader Shows That Title and Content Are Missing**

For example, when an entity is exposed that contains a Title property (like our Post entity), it's mapped to the Atom "title" element.

On the other hand, if there's no automatic mapping, you can override that behavior by providing an explicit mapping based on the entity name, as we've done to map the PublishDate property for objects in the "Posts" collection to the "published" atom property. Those two attributes are enough to turn our OData feed into an Atom feed, giving us the full-featured view of the data as shown in **Figure 7**.

This mapping is not one-way; it supports all of the HTTP methods, so you can use the AtomPub protocol to create, up-

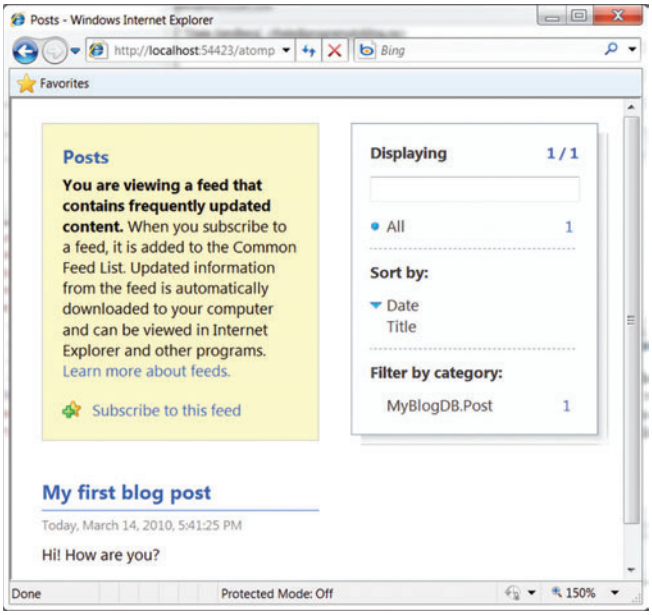
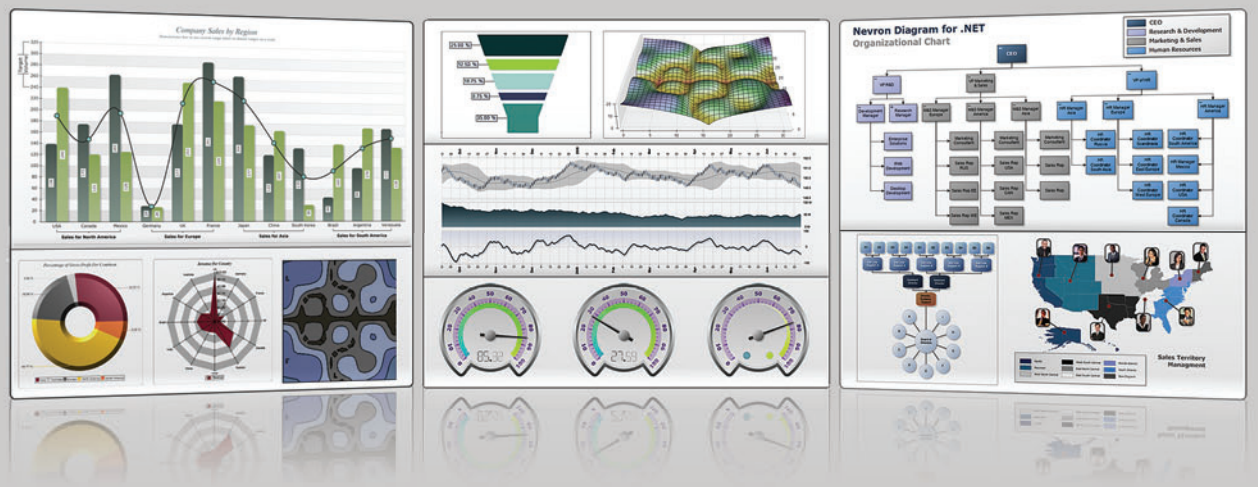


Figure 7 **Tweaking the XML Format Results in Correct Display of the Title and Content**

NEVRON
visualize your success

Let us help you visualize your success

Nevron provides the essential components for the creation of advanced digital dashboards, scientific and financial applications, diagrams and MMI interfaces for a variety of .NET centric technologies.



Nevron components integrate seamlessly in web and desktop applications, SQL Server Reporting Services 2005/2008 reports and SharePoint 2007/2010 portals and deliver an unmatched set of enterprise-grade features. That is why Nevron is the trusted vendor by many Fortune 500 companies for their most demanding data visualization needs.

Developers

Nevron .NET Vision incorporates components that help you create enterprise grade digital dashboards, scorecards, diagrams, maps, MMI interfaces and much more.

- Chart for .NET
- Diagram for .NET
- Gauge for .NET
- Map for .NET
- User Interface for .NET

IT Professionals

Nevron Reporting Services Vision instantly enhances your SQL Server Reporting Services 2005/2008 reports with the industry leading data visualization technology.

- Chart for SSRS
- Gauge for SSRS

Nevron SharePoint Vision instantly converts your SharePoint pages into advanced dashboards and reports, that unite powerful data analysis with industry leading data visualization.

- Chart for SharePoint
- Gauge for SharePoint

MAKE SURE THAT YOUR DATA IS MAKING THE VISUAL STATEMENT IT DESERVES BY DOWNLOADING YOUR FREE EVALUATION COPY FROM WWW.NEVRON.COM TODAY.

Microsoft .NET | ASP.net AJAX | Microsoft SharePoint 2007/2010 | Microsoft SQL Server 2005/2008 | Microsoft Visual Studio PARTNER

www.nevron.com | sales@nevron.com | 1888 201 6088

Some Nevron components only available for certain platforms. For details visit www.nevron.com or send an email to support@nevron.com.

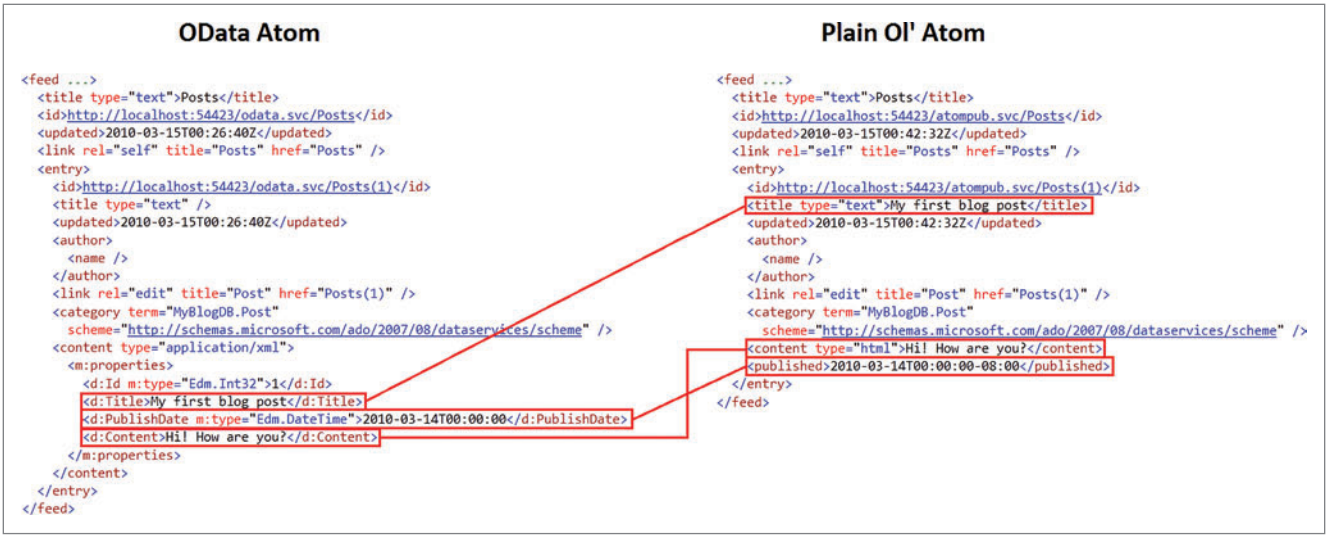


Figure 8 **Mapping Between Atom and OData**

date and delete items in the Posts collection as well as read them. This means you can configure a tool like Windows Live Writer (WLW), which supports AtomPub as a blog API, and use it for rich text editing of your posts. For example, given the atompub.svc endpoint, in WLW, you could choose Blogs | Add blog account and fill in the following options in the dialogs that follow:

- What blog service do you use? Other blog service
- Web address of your blog: http://<<server>>:<<port>>/atompub.svc
- Username: <<username>> (required and should be implemented on your AtomPub endpoint using standard HTTP techniques)
- Password: <<password>>
- Type of blog that you're using: Atom Publishing Protocol
- Service document URL: http://<<server>>:<<port>>/atompub.svc
- Blog nickname: <<whatever you like>>

Click Finish and you've got a rich text editor for managing your blog posts, as shown in **Figure 9**.

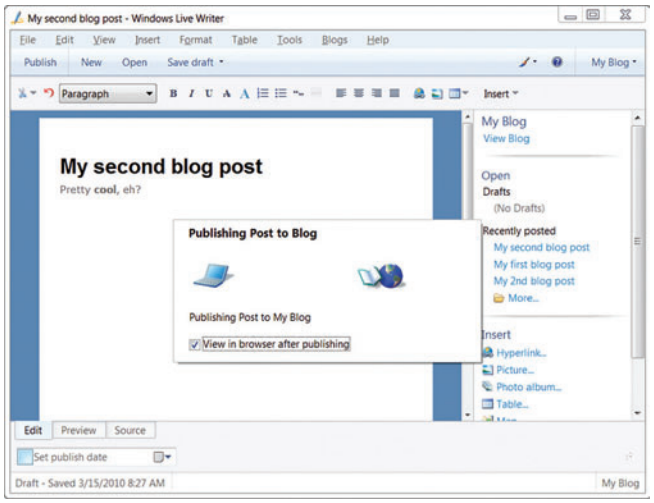


Figure 9 **Atom/OData Mapping Facilitates Building a Rich Text Editor to Manage Your Blog Posts**

Here we've taken the Data Services engine, which supports full CRUD functionality by packing properties into the Atom "content" element, and have done a little bit of mapping to make it support plain ol' Atom and AtomPub, too.

The little sample library that I used to make this work (which I built with Phani Raj, a software engineer at Microsoft on the WCF Data Services team), does the barest minimum and it's never going to be all you need to build a real blog. Here's a list off the top of my head of things it still needs to really support just Atom and AtomPub:

> The goal is to make the OData feeds friendly, not abandon OData in favor of Atom/AtomPub.

- Mapping the Atom author element sub-elements, such as name, uri and e-mail.
- Handling images (although WLW allows for FTP, so that might be enough).
- Exposing capabilities to make WLW recognize these features.

If you're interested in pushing this experiment further, Joe Cheng, a member of the WLW team, has written a series of blog posts about AtomPub support in WLW that inspired this work in the first place: jcheng.wordpress.com/2007/10/15/how-wlw-speaks-atompub-introduction.

Enjoy! ■

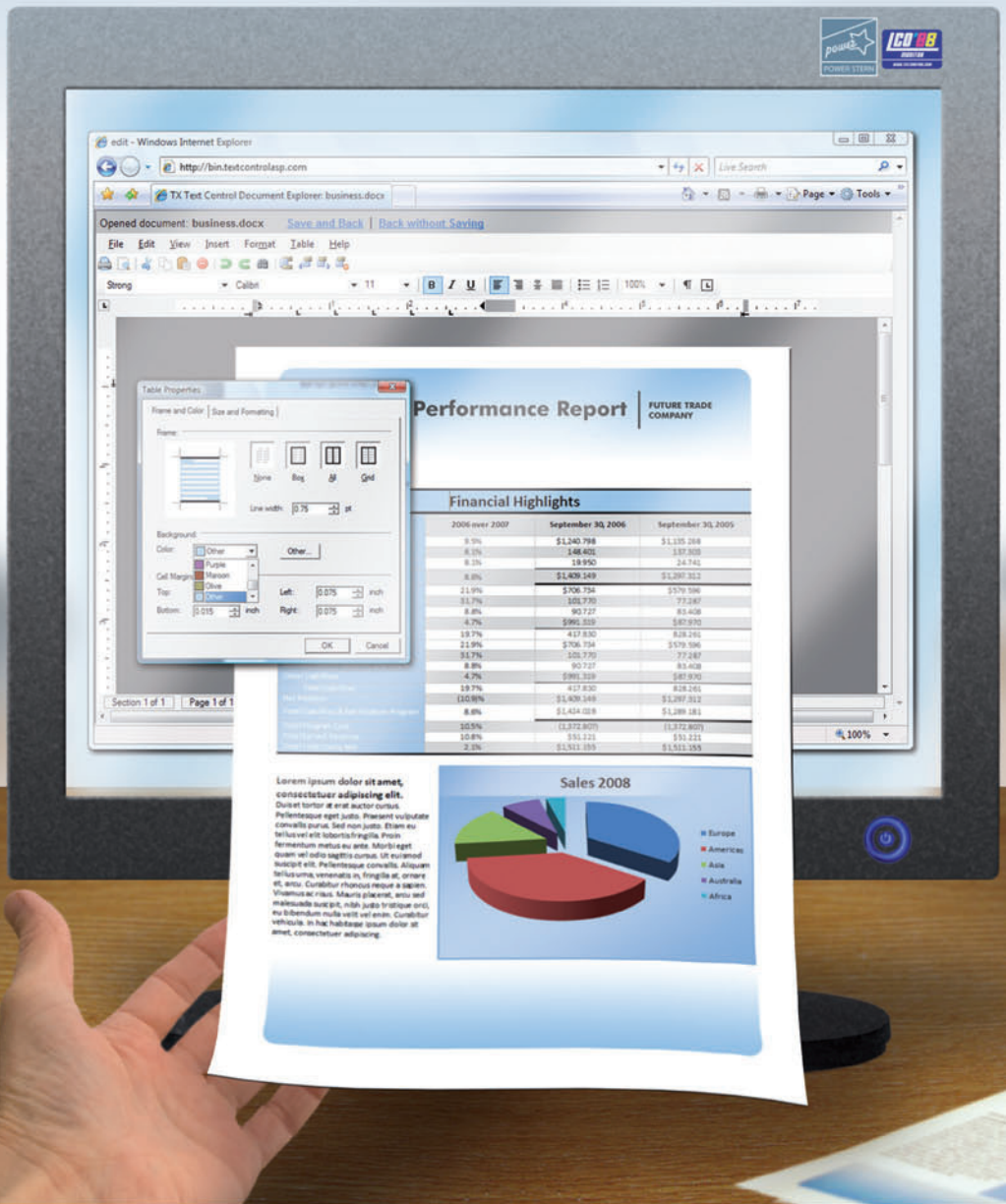
CHRIS SELLS *is a Microsoft program manager in the Business Platform Division. He's written several books, including the coauthored "Programming WPF" (O'Reilly Media, 2007), "Windows Forms 2.0 Programming" (Addison-Wesley Professional, 2006) and "ATL Internals" (Addison-Wesley Professional, 1999). In his free time, he hosts various conferences and makes a pest of himself on Microsoft internal product team discussion lists. More information about Sells, and his various projects, is available at sellsbrothers.com.*

THANKS to the following technical expert for reviewing this article: *Pablo Castro*

WINDOWS FORMS / WPF / ASP.NET / ACTIVEX

WORD PROCESSING COMPONENTS

(WHAT YOU SEE IS WHAT YOU GET)



TX TEXT CONTROL® word processing components

Word Processing Components
for Windows Forms & ASP.NET

www.textcontrol.com

Microsoft® Visual Studio®
PARTNER

US +1 877 - 462 - 4772 (toll-free)
EU +49 421 - 4270671 - 0

Tuning Your Database Calls with Tier Interaction Profiling

Mark Friedman

Many applications are designed explicitly to use multiple tiers, where the performance of calls into the data access tier is crucial to the overall responsiveness of the application. The use of multiple tiers increases the flexibility of the application. The n-tiered approach also fosters isolation of key components, which can be used to improve both reliability and scalability. The use of multiple tiers promotes scalability because components that are isolated into distinct tiers can then more readily be distributed across available computing resources.

Tier interaction profiling (TIP) is designed to help you understand the performance of the data layer on which the application relies. TIP is a new feature of the Visual Studio profiling tools that measures and reports the duration of data tier delays .NET Framework applications experience when waiting for synchronous calls to ADO.NET-compliant databases to complete. For response-time-oriented applications that make frequent calls to databases, TIP helps you to understand which data requests contribute the most to response delays.

In this article, I'll introduce you to TIP and demonstrate its reporting capabilities. I'll also discuss the instrumentation technology that TIP relies on and provide some best practices for using TIP effectively to diagnose performance problems related to database activity. I will walk you through using TIP on a sample two-tiered ASP.NET Web application that is data-intensive and that accesses data from Microsoft SQL Server using the LINQ to SQL technology. Finally, I will discuss how you can augment the TIP performance data by using standard SQL Administrator performance tools to gain an even deeper understanding of the performance of a data tier.

This article discusses:

- Introduction to Tier Interaction Profiling (TIP)
- Using TIP to improve the performance of applications
- Reading a TIP Report
- TIP data with LINQ to SQL
- Choosing a primary profiling method

Technologies discussed:

ADO.NET, Visual Studio, SQL Server

Getting Started with TIP

TIP dynamically adds instrumentation code to measure the duration of calls to the data layer of your application during a profiling run. The Visual Studio profiling tools are available in Visual Studio 2010 Premium Edition and Visual Studio 2010 Ultimate Edition.

To begin a profiling session, you can either click Launch Performance Wizard from the Analyze menu, click Start Performance Analysis from the Debug menu, or use the Alt+F2 keyboard shortcut. The first page of the Performance Wizard then asks you to select a profiling method.

TIP can be used with any profiling method (sampling, instrumentation, memory, or concurrency), but it isn't enabled by default. To enable TIP, you will need to uncheck the "Launch profiling after the wizard finishes" checkbox on page 3 of the Performance Wizard. (Because TIP isn't turned on yet, you're not ready to launch your application and begin profiling.) For best results, I recommend selecting the sampling method of profiling initially, especially if what you care about most is the data tier interaction data. This is mainly because sampling has least impact on the performance of your application.

To enable TIP data collection, access the Performance Session in the Performance Wizard just created in the Performance Explorer window and right-click to see its properties. In the properties dialog, select the Tier Interactions tab, then check the "Enable tier interaction profiling" checkbox. Click the OK button to dismiss the dialog. Now that TIP is enabled, you're ready to start profiling your application.

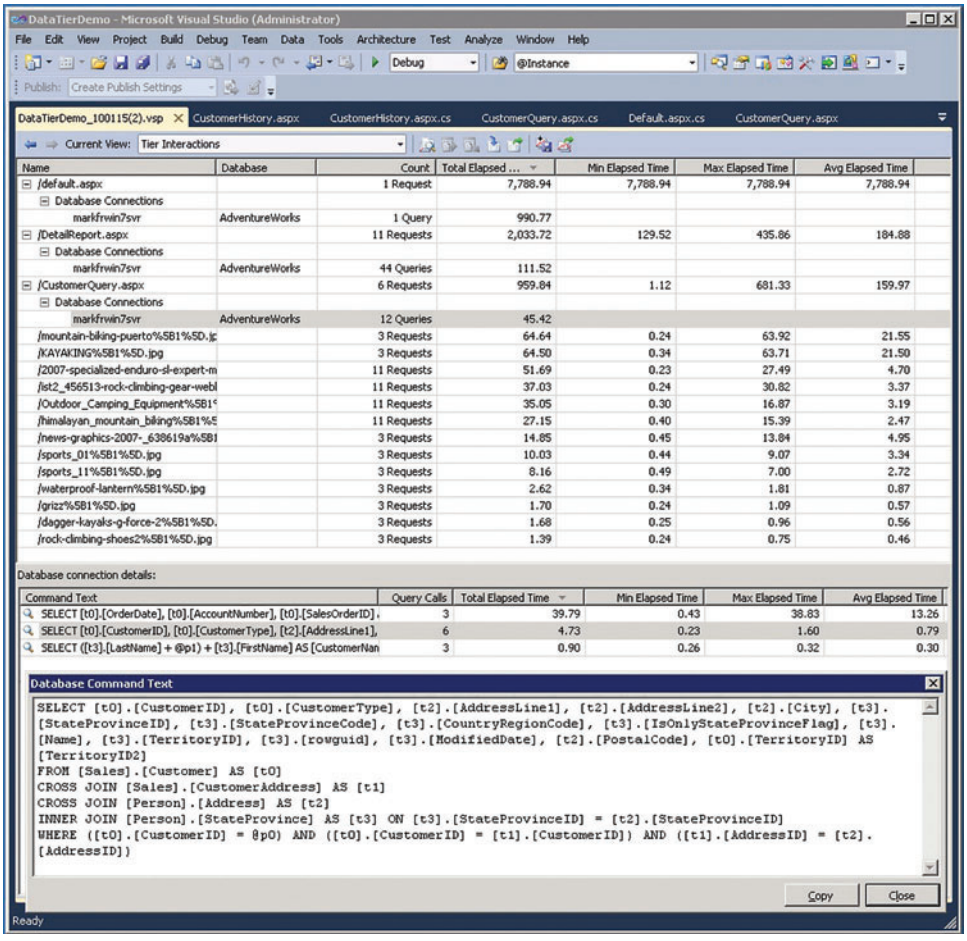


Figure 1 **A Visual Studio Profiler Tier Interaction Report**

To start the actual profiling run, click the Launch with Profiling toolbar button in the Performance Explorer window

For complete instructions on enabling TIP in Visual Studio, see Habib Heydarian's blog post "Walkthrough: Using the Tier Interaction Profiler in Visual Studio Team System 2010" at blogs.msdn.com/b/habibh/archive/2009/06/30/walkthrough-using-the-tier-interaction-profiler-in-visual-studio-team-system-2010.aspx.

How TIP Measures Performance

TIP inserts code into your application during a profiling run that measures calls to the ADO.NET data layer your application uses. When tier interaction profiling is active, the Visual Studio Profiler inspects the Microsoft intermediate language (MSIL) in your solution, looking for references to ADO.NET functions. Prior to calling the just-in-time (JIT) compiler to generate the native code run by the application, the profiler inserts instructions that add instrumentation to key ADO.NET methods. This instrumentation code keeps track of the amount of time spent during each ADO.NET call.

As the application executes, TIP captures and records timing data. When your application is being profiled, this instrumentation records the duration of any ADO.NET calls that are made and also captures a copy of the command text used in the database call. As your application runs, timing data is gathered for all database accesses performed using synchronous methods of the ADO.NET classes, including SQL, OLE DB, Open Database Connectivity (ODBC), and SQL Server Compact (SQL CE) APIs. TIP also captures timing data if your application uses LINQ to SQL or the Entity Framework APIs to access SQL databases.

> TIP helps you to understand which data requests contribute the most to response delays.

The timing data is stored along with any other data gathered during the profiling session in a Visual Studio profiler file (.vsp). Because an application making a call to an external database performs an out-of-process call, the instructions that TIP adds to instrument the application have very little impact on overall application performance.

Tuning for Performance

A common design pattern involves dividing a Web application into a presentation layer, a business logic layer and a data layer. This

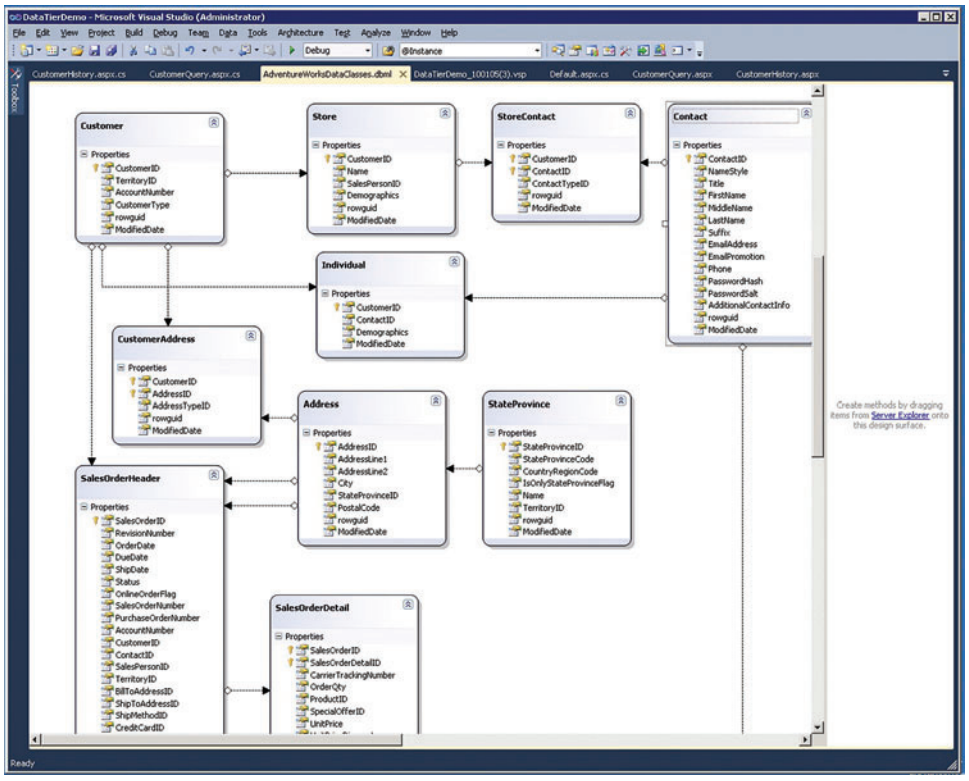


Figure 2 **The AdventureWorks Tables Used to Query Sales.Customer Information**

design paradigm leads to breaking your application into component parts to foster reliability, extensibility and scalability. The multi-tiered application accesses its data layer using business logic components that reference data entities logically as rows and columns in a related set of tables. How a database like SQL Server maintains the physical data associated with the database tables is transparent to your application by design.

> TIP data lets you drill down into database-related delays and understand why they're occurring.

For the sake of reliability and scalability, large-scale Web applications often configure multiple machines into pools that are responsible for the processing logic associated with each layer of the application. Multiple machines supporting multiple tiers creates a special challenge in performance analysis because monitoring any single machine provides an incomplete picture of the application.

For example, using a database system like SQL Server to manage and mediate access to the application's data store creates a data layer that is isolated from the application logic. The application data housed in a database like SQL Server is maintained in a separate process address space with its own data store. The database containing the application data can reside on the same physical machine as the application, but is more likely to be accessed from a different machine using networking protocols.

SQL commands that the application passes to an external database and that are operated upon there are out-of-process calls. Sampling profiles see that the application is dormant while waiting for these out-of-process calls to complete, but cannot tell you why the application is waiting or the duration of these delays. Instrumented and concurrency profiles do measure the duration of these delays, but cannot tell you what SQL commands are being issued or why they're taking so long to complete.

In multi-tiered applications that communicate with an external database, the database component is often a major contributor to overall application response time. The Visual Studio profiling tools include a variety of profiling methods including sampling, instrumentation, memory allocation, and concurrency—but none of these methods may be of much help in identifying performance issues associated with accessing an external database without TIP data.

TIP data lets you drill down into database-related delays and understand why they're occurring. In conjunction with other performance tools your database vendor can supply, you can also start to understand what can be done to improve the performance of an application that depends heavily on database performance.

Because TIP adds instrumentation to the application code, timing data associated with database commands can be gathered no matter where the database instance being accessed is physically located. Timing data can be gathered, for example, for out-of-process calls to a SQL Server instance physically resident on the same machine as the application, which is a typical scenario in unit testing. When the same application is ready for integration or load testing against a SQL Server instance located on a separate physical machine, TIP can continue to gather measurement data for that configuration. In fact, the TIP measurements enable you to compare the performance of these two different configurations.

TIP lets you compare and contrast the impact of the many external database performance and tuning options that are available, including cache memory configuration, physical data storage devices, database partitioning, database indexing and database table design. In addition, you can directly measure the performance impact of running SQL Server on a virtual machine.

TIP Report Basics

Once a profiling session with TIP activated completes, the timing data associated with any of your application's interaction with its ADO.NET data layer is summarized in the Tier Interactions view. **Figure 1** shows an example of the profiler when TIP data gathering is active and there is ADO.NET activity during the profiling run.

The top half of the report is a summary of the profiling data that was gathered. For ASP.NET applications, the view is organized by URL. The report groups server-side response time of Web application GET requests by URL.

Below the application layer, the report shows each connection to a database tier (which in this example was the AdventureWorks sample database). It measures and reports the portion of the server-side processing time for ASP.NET requests that is associated with synchronous database calls using ADO.NET. In this example three summary lines appear, each tabulating the database activity associated with three different ASP.NET pages in the Web site being profiled. For each ASP.NET page identified during profiling, the number of ASP.NET requests that were processed during the profiling run and the server-side response times for each response message that was generated are reported.

> TIP can be especially helpful with applications that use LINQ to SQL.

Additional summary lines show response-time data for other GET requests, including requests for stylesheets, Javascript code, and images linked in the pages. Any database calls that the profiler is unable to associate with a specific ASP.NET request are grouped under an Other Requests category.

When you're profiling a Windows desktop or console application that utilizes a data layer, the report organizes the ADO.NET activity under the process name.

Below each Web page summary line is a single summary line that reports the number of synchronous database requests, organized by database connection, that were made during ASP.NET processing. In this example, you can see that six ASP.NET requests to CustomerQuery.aspx were handled over a single database connection. These six requests took a total of 0.959 seconds to process on the server, for an average response time of 160 milliseconds. These requests issued 12 SQL queries that took approximately 45 milliseconds to complete. Waiting for the database requests accounted for only about 5 percent of the elapsed time associated with generating the response messages for this Web page.

If you highlight one of the database connection summary lines, the bottom half of the Tier Interactions view breaks out the specific SQL commands that the application issued. The SQL commands are grouped by the command text issued and sorted by elapsed time within the page group.

In the example, one SQL command was issued three times, another command was issued six times, and a third query was issued three times. In the detail view, the elapsed time of each specific query that is rolled up into a single line on the summary report is reported separately. You can see the total elapsed time, the average across all instances of the command, and the minimum and maximum delays that were observed for each query.

If you double-click the SQL command detail line, the full text of the SQL command that was issued is displayed in a Database Command Text window. This is the actual command the application passed to the database across the ADO.NET interface during execution. If the request is for execution of a stored procedure, the specific call to the stored procedure is shown.

Figure 3 **LINQ to SQL Customer Query**

```
var customerquery =
  from customers in db.Customers
  from custaddrs in db.CustomerAddresses
  from addrs in db.Addresses
  where (customers.CustomerID == custid &&
         customers.CustomerID == custaddrs.CustomerID &&
         custaddrs.AddressID == addrs.AddressID)

  select new {
    customers.CustomerID,
    customers.CustomerType,
    addrs.AddressLine1,
    addrs.AddressLine2,
    addrs.City,
    addrs.StateProvince,
    addrs.PostalCode,
    customers.TerritoryID
  };
```

A LINQ to SQL Example

Let's look at a simple example of using TIP to understand an ASP.NET application that depends heavily on accessing information from a database.

TIP can be especially helpful with applications that use LINQ to SQL to access data stored in an external SQL Server database because LINQ tends to remove the developer one step further from the physical database and its performance characteristics. With LINQ to SQL, the Entity:Relationship (E:R) diagrams that you create in the Object Relational Designer generate classes that are then used by Visual Studio as templates for building syntactically correct SQL commands automatically.

Because LINQ to SQL eliminates most SQL language coding considerations and is easy to use, it also tends to mask important performance considerations associated with database design, configuration and tuning. As this example illustrates, using LINQ,

Figure 4 **SQL Command for customerquery**

```
SELECT [t0].[CustomerID], [t0].[CustomerType], [t2].[AddressLine1],
[t2].[AddressLine2], [t2].[City], [t3].[StateProvinceID],
[t3].[StateProvinceCode], [t3].[CountryRegionCode], [t3].
[IsOnlyStateProvinceFlag], [t3].[Name], [t3].[TerritoryID], [t3].
[rowguid], [t3].[ModifiedDate], [t2].[PostalCode], [t0].[TerritoryID] AS
[TerritoryID2]
FROM [Sales].[Customer] AS [t0]
CROSS JOIN [Sales].[CustomerAddress] AS [t1]
CROSS JOIN [Person].[Address] AS [t2]
INNER JOIN [Person].[StateProvince] AS [t3] ON [t3].[StateProvinceID] =
[t2].[StateProvinceID]
WHERE ([t0].[CustomerID] = @p0) AND ([t0].[CustomerID] = [t1].
[CustomerID]) AND ([t1].[AddressID] = [t2].[AddressID])
```

you can easily create a complex query that joins multiple tables without having to consider the performance implications of doing so.

With TIP, you can see the actual SQL command text that LINQ to SQL generates and gather measurements from the runtime execution of these SQL queries. With the SQL command text in hand, you can then access other database tuning tools to help you better understand the performance implications of any particular LINQ to SQL operation.

> Using TIP, you can start to understand the performance implications of these seemingly simple LINQ to SQL operations.

My example is a Web Form application that queries the AdventureWorks Sales.Customer tables using a specific customer ID to retrieve that customer's order history. The AdventureWorks tables involved in this query include the Customer, SalesOrderHeader, SalesOrderDetail and StateProvince tables, as illustrated in the Object Relational Designer view in **Figure 2**.

If you also wanted to display the customer mailing and e-mail address information alongside the order history, the CustomerAddress, Address, and Contact tables need to be accessed. As shown in the Object Relational Designer, the AdventureWorks tables contain primary and foreign keys such as CustomerID, SalesOrder, and ContactID that allow these tables to be joined logically.

The C# code to create an AdventureWorks customer query using LINQ to SQL is shown in **Figure 3**. In this case custid is the specific CustomerID value requested. This query returns a customeryquery collection that contains a single data row containing the data fields listed in the select new clause.

The customeryquery can then be bound to a control on an ASP.NET Web page:

```
DetailsView1.DataSource = customerquery;
DetailsView1.DataBind();
```

Now I can create a query to retrieve the order history of this customer:

```
var orderquery =
  from orderhdr in db.SalesOrderHeaders
  where (orderhdr.CustomerID == custid)
  orderby orderhdr.OrderDate
  select new {
    Date = orderhdr.OrderDate.ToShortDateString(),
    orderhdr.AccountNumber,
    InvoiceNo = orderhdr.SalesOrderID,
    orderhdr.TotalDue
  };
```

When this LINQ to SQL operation executes, orderquery will contain a row corresponding to each row in the OrderHdr table associated with a specific Customer ID. The orderquery collection will contain multiple rows if the customer history indicates there were multiple sales transactions.

These queries look straightforward on the surface. But using TIP, you can start to understand the performance implications of these seemingly simple LINQ to SQL operations.

Using TIP Data for Tuning

Let's take a closer look at customerquery. At run time, LINQ to SQL takes the logical database SELECT operations implied in the LINQ statement and uses it to generate a valid SQL command that joins data from four AdventureWorks tables: Customers, CustomerAddresses, Addresses and the static StateProvince table. You don't see this in the LINQ to SQL code here.

When you run this code under the Visual Studio profiler, the TIP instrumentation reports the number of times this query was executed and measures the time the Web page was delayed waiting for its execution. This is, in fact, the operation that executed six times during the profiling run illustrated back in **Figure 1**.

In addition, as you saw earlier, the SQL command generated by the LINQ to SQL code is also available when the application is profiled. **Figure 4** shows the actual SQL command for this operation. Note that the SQL command text includes a token (designated here as "@p0") to represent the customer ID parameter that LINQ feeds into the query.

Now that the actual SQL command text generated by LINQ is

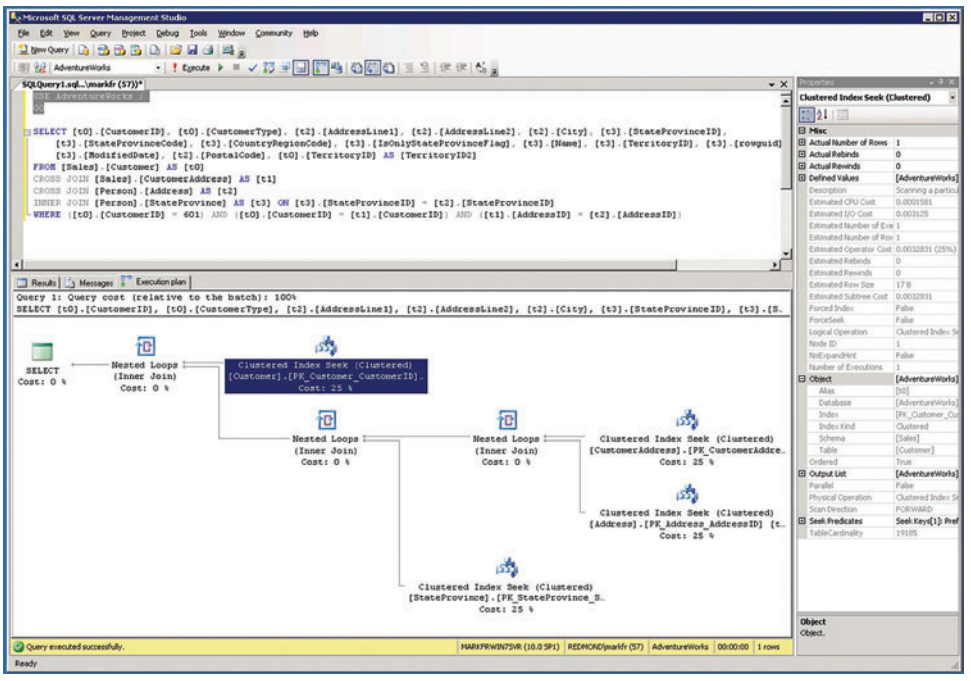


Figure 5 **The Execution Plan for the Example LINQ to SQL Operation**

WindwardReports

HOW EPIC IS WINDWARD REPORTS' EARTH-SHATTERING APPROACH TO CUSTOM ENTERPRISE REPORTING?

It's so powerful it makes the Kraken look like a gimp sea monkey.

It's so fast it makes warp drive look like a hobbit running backwards.

IT'S SO EASY IT MIGHT AS WELL BE "GOD MODE"

It makes other reporting solutions seem like you're trying to crack **RIJNDAEL ENCRYPTION**, or like driving the "**ROAD OF DEATH**" in the Bolivian Andes, backwards, while blindfolded.

No other solution can match Windward's array of features, **STREAMLINED IMPLEMENTATION**, and already familiar interface. You create custom report templates with Word, Excel, or PowerPoint. Even your co-workers **who need IT to turn on their computers** can use Microsoft Office to format reports.

Windward Reports hasn't simply shifted the reporting paradigm,

IT HAS NINJA ROUND HOUSE KICKED IT ALL THE WAY INTO AN ENTIRELY DIFFERENT DIMENSION.

IF YOU FIND ALL OF THIS HARD TO SWALLOW,

DOWNLOAD THE FREE TRIAL AT

www.WindwardReports.com/msdn.aspx

AND SEE FOR YOURSELF. **

- **Design Reports in Microsoft Word, Excel, and PowerPoint.**
- **Drag N'Drop data into report templates *no coding required!***
- **Solutions for .Net, Java and SharePoint platforms**
- **Integrates easily into any software application**

the ONLY **EPIC** REPORTING & DOCUMENT GENERATING SOLUTION

** Unless of course you enjoy designing report templates with endless code, apologies for keeping you from **your current mind-numbingly dull solution.**

(303) 499-2544

available, it's possible to understand how the database design is affecting the performance of the query.

One thing you can do is execute this SQL command in SQL Server Management Studio and examine its execution plan, as shown in **Figure 5**. To access the execution plan for this query, you need to add a command to point to the appropriate database:

```
USE AdventureWorks ;
GO
```

Next, copy the SQL command text from the TIP report, remembering to replace the "@p0" token with a valid CustomerID from the database. Then execute this example query in SQL Server Management Studio and access the execution plan that shows how the query optimizer translated the logical request into a physical plan of execution.

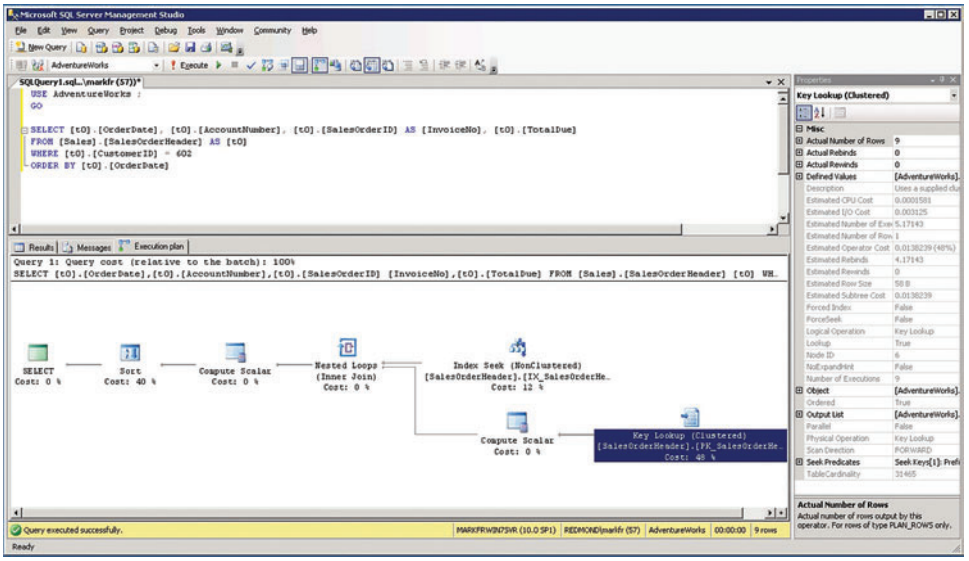


Figure 6 **The Execution Plan for orderquery**

In this example, the execution plan for the query shows that the SELECT statement accessed the Customer table using a clustered index on the CustomerID field that returned exactly one row in the table. In SQL Server Management Studio you can use the mouse to hover over an operation to view its properties or highlight the operation and right-click to view the Properties window. In this fashion, you can cycle through each of the remaining operations that the command requests. Each of the three subsequent JOINs that augment the initial Customer SELECT also accesses a table using its clustered index and returns a single unique row.

From this investigation, you can see that, to process this query, a total of four rows, each from a different table in the AdventureWorks database, needs to be accessed. Each access is performed efficiently utilizing the table's unique primary key.

> TIP data can be gathered during sampling, instrumentation and concurrency profiling runs.

Similarly, you can use TIP to see the SQL command for the orderquery code and feed it to SQL Server Management Studio to see its execution plan (see **Figure 6**). This query accesses a single table called OrderHdr using the CustomerID as a foreign key, so it needs to access both an ordinary nonclustered index as well as the clustered index on SalesOrderHeaderID.

This particular instance of the query returns nine rows. The orderby clause in the LINQ to SQL code translates into a SQL ORDER BY clause that entails an additional Sort operation on the resultset of the SELECT. This operation represents 40 percent of the overall cost of executing the request, as estimated by SQL Server Plan Optimizer.

Choosing a Profiling Context

TIP is designed to supplement existing Visual Studio profiling methods to gather specific measurements on data tier interactions. TIP is a secondary data-gathering facility and cannot be gathered without specifying a primary profiling method. TIP data can be gathered during sampling, instrumentation and concurrency profiling runs for any application that communicates with a data tier using ADO.NET.

Given that you need to choose a primary profiling method for an application for which you want to gather TIP data, which should you use? Let's walk through some of the considerations for choosing a primary profiling method.

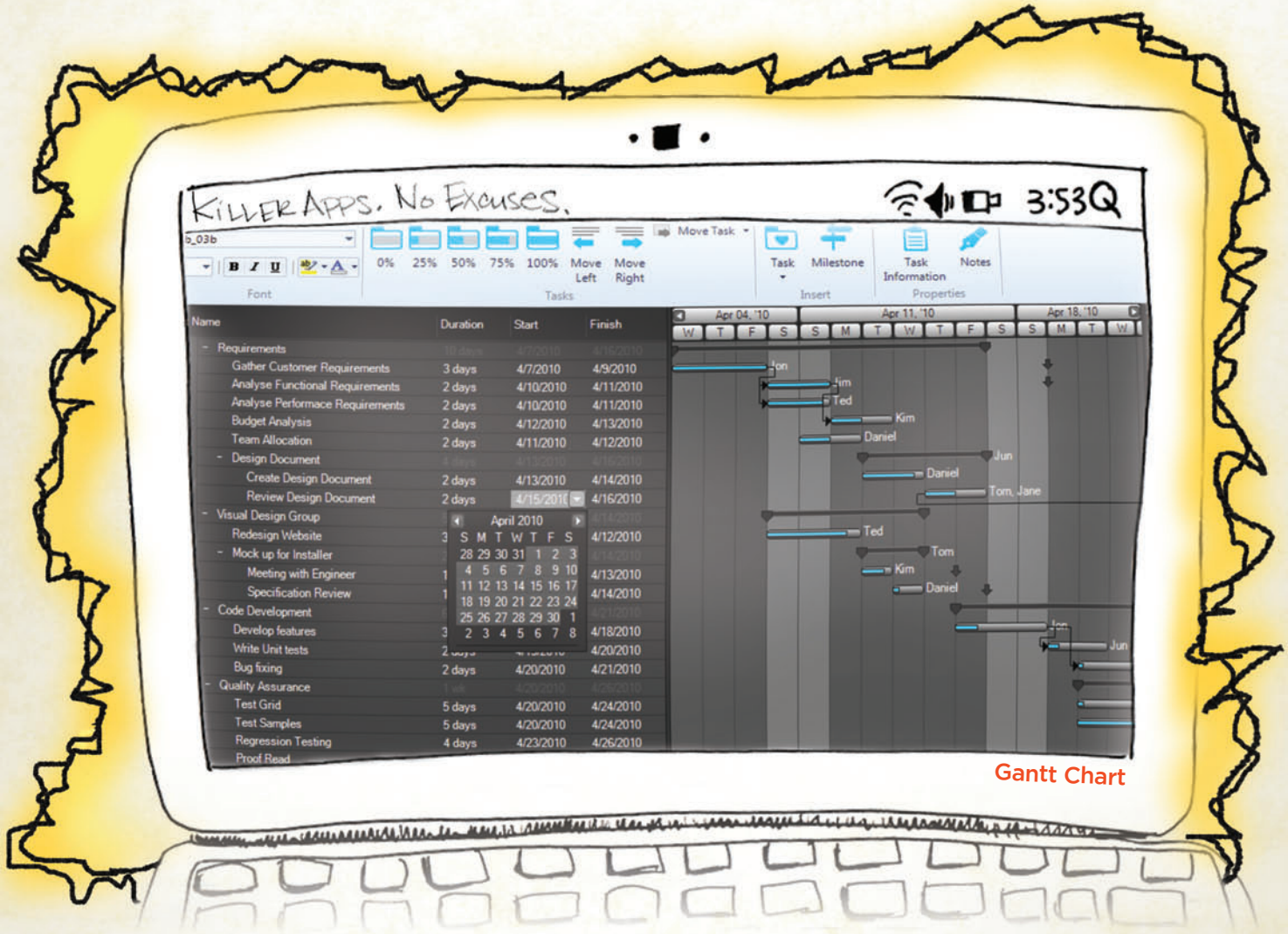
Are the delays associated with data tier interactions the primary focus of the performance investigation? If so, sampling profiling is the recommended primary method because it's usually the least intrusive form of profiling.

If data tier delays aren't the primary focus of the performance investigation, choose a profiling method based on what is most likely to provide the measurement data that is most applicable in the current context. For instance, if you're investigating a problem associated with the concurrent execution of multiple threads, gather concurrency data. If you're investigating a problem associated with a CPU-bound application, gather sampling data. For additional guidance on which primary collection method to choose, see the article "How to: Choose Collection Methods" (msdn.microsoft.com/library/ms182374(VS.100)).

If you're not already familiar with the data layer code, you may need help from the primary profiling data to find the exact code where these ADO.NET calls originate. TIP does not capture a call stack when it gathers timing information on synchronous ADO.NET out-of-process calls. If you need to understand where in the application calls to ADO.NET methods are being made, instrumented profiles will be the most helpful. Sampling data can also help you do this, but not with the precision of an instrumented profile.

You can choose to gather resource-contention data along with the tier interaction measurements, but gathering contention data

INNOVATION TO SPARK KILLER APPS



Gantt Chart

You have the vision, but time, budget and staff constraints prevent you from seeing it through. With rich user interface controls like Gantt Charts that Infragistics **NetAdvantage® for .NET** adds to your Visual Studio 2010 toolbox, you can go to market faster with extreme functionality, complete usability and the "Wow-factor!" Go to **infragistics.com/spark** now to get innovative controls for creating Killer Apps.

Infragistics®
KILLER APPS. No Excuses.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111
twitter.com/infragistics

tends to be a higher-overhead function than sampling, and the contention data is unlikely to provide any assistance in helping to determine where specific ADO.NET calls originate. Investigations that require .NET Memory Allocation profiling, which are normally very high impact, are unlikely to benefit from gathering tier interaction measurements.

Sampling Profiles

Often, the data tier interactions themselves are the primary focus of a performance investigation. In this case, best results are usually obtained when sampling is chosen as the primary profiling method. Sampling is preferred in this case mainly because it's the profiling method that usually affects an application's performance the least. Sampling profiles can also prove helpful in navigating to the source code where the ADO.NET calls that most affect performance originate.

> Tier Interaction Profiling explicitly identifies and measures out-of-process delays separately.

With data tier functions that run out-of-process, the instruction execution samples gathered during a sampling profile normally do not reflect any of the time that an application spends waiting for synchronous calls across an ADO.NET interface to complete. During periods when an application's execution thread is waiting for these out-of-process calls to complete, the application thread is blocked and no execution samples are recorded against it. When sampling is used, gathering TIP data is the best way to understand which application delays occur that are due to synchronous calls to a data tier.

The instrumentation used by TIP doesn't capture call stacks when it gathers timing data. So, if you're profiling a tiered application and aren't thoroughly familiar with the code, it can be difficult to determine precisely where the calls to the data layer originate. Sampling profiles may also help identify where in the application code calls to these ADO.NET interfaces are performed. If the application makes frequent calls to ADO.NET interfaces, some samples showing time spent in ADO.NET modules, including System.Data.dll and System.Data.Linq.dll, are likely to be gathered.

In reviewing the sampling data and comparing it to the tier interaction measurements, keep in mind that while an application thread is blocked waiting for a synchronous database call to complete, no sampling data for the thread is gathered. The samples accumulated during execution exclude the out-of-process delays TIP explicitly measures. What you can expect, however, is a rough correlation between the execution samples gathered in ADO.NET methods and the number of ADO.NET commands that TIP observes and measures. In such cases, sampling profiles can assist you in navigating to the source code where the ADO.NET calls that TIP measures and reports originate.

Note that if an application has SQL queries that return large resultsets that are then bound to a data-bound control on a form, you're apt to find a significant number of execution samples in the control's DataBind method. Seeing which DataBind methods appear in the sampling profile may also assist you in navigating to the source code where the ADO.NET calls originate.

Instrumented Profiles

When instrumented profiles are gathered, the timing data for methods recorded by the instrumentation already include any time spent waiting for out-of-process calls to complete inside the method. The timing data recorded in an instrumented profile is gathered by measuring the time at the entry and exit of each of the application's methods that are selected for instrumentation. Timing data for methods in an application that interface to a data-tier layer using ADO.NET calls implicitly include the delay in executing any out-of-process calls.

The timing data gathered from TIP explicitly identifies and measures out-of-process delays separately. The delay measured by tier interaction profiling should be a subset of the overall time spent inside the method, as measured during an instrumented profiling run. With this understanding, you should be able to match the timing data from tier interaction profiling against the timing data gathered at the method level in an instrumented profile to pinpoint the method where the call to the data layer originated.

If instrumentation at the method level is sufficient to allow you to figure out where in the application any ADO.NET calls originate, then instrumented profiling can be used without hesitation. However, instrumented profiles are usually much more intrusive than sampling profiles, incur greater overhead, and tend to generate significantly larger .vsp collection files. In addition, if your application uses methods that make multiple calls to ADO.NET functions, the data gathered by instrumentation will only help you navigate to the method level, not differentiate among multiple ADO.NET calls embedded inside a single method.

More Data, Please

Building multi-tiered applications is a design pattern that promotes reliability and scalability, but does lead to performance-monitoring challenges when application components execute on different machines.

A simple view of a multi-tiered application that doesn't encompass all of its interconnected layers can't provide a complete picture of its performance. As you've seen, TIP can supply key timing data that's otherwise missing. As the example in this article suggests, this timing data can be augmented with other performance data from standard data base administrator tools. ■

Mark Friedman *is an Architect on the Visual Studio Ultimate team at Microsoft. He's the author of two books on Windows performance and blogs periodically about performance issues at blogs.msdn.com/ddperf/.*

Thanks to the following technical experts for reviewing this article: *Daryush Laqab and Chris Schmich*

Building Distributed Apps with NHibernate and Rhino Service Bus, Part 2

Oren Eini

In the July 2010 issue of *MSDN Magazine*, I started walking through the process of building a smart client application for a lending library. I called the project Alexandria, and decided to use NHibernate for data access and Rhino Service Bus for reliable communication with the server.

NHibernate (nhforge.org) is an object-relational mapping (O/RM) framework, and Rhino Service Bus (github.com/rhino-esb/rhino-esb) is an open source service bus implementation built on the Microsoft .NET Framework. I happen to be deeply involved in developing both of these frameworks, so it seemed like an opportunity to implement a project with technologies I know intimately, while at the same time provide a working example for developers who want to learn about NHibernate and Rhino Service Bus.

In the previous article, I covered the basic building blocks of the smart client application. I designed the back end, along with the communication mode between the smart client application and the back end. I also touched on batching and caching, how to manage transactions and the NHibernate session, how to consume and reply to messages from the client, and how everything comes together in the bootstrapper.

In this installment, I will cover best practices for sending data between the back end and the smart client application, as well as patterns for distributed change management. Along the way, I'll cover the remaining implementation details, and will present a completed client for the Alexandria application.

You can download the sample solution from github.com/ayende/alexandria. The solution comprises three parts: Alexandria.Backend hosts the back-end code; Alexandria.Client contains the front-end code; Alexandria.Messages contains the message definitions shared between them.

This article discusses:

- Commands over change sets
- Local state management
- Back-end processing
- Handling complex operations

Technologies discussed:

NHibernate, Rhino Service Bus

No One Model Rules

One of the most common questions people ask when writing distributed applications is: How can I send my entities to the client application and then apply the change set on the server side?

If that's your question, you're probably thinking in a mode where the server side is mostly a data repository. If you build such applications, there are technology choices you can make that simplify this task (for example, employing WCF RIA Services and WCF Data Services). Using the type of architecture I've outlined so far, however, it doesn't really make sense to talk about sending entities on the wire. Indeed, the Alexandria application uses three distinct models for the same data, each model best suited for different parts of the application.

The domain model on the back end, which is used for querying and transactional processing, is suitable for use with NHibernate (and further refinement would be to split the querying and transactional processing responsibilities). The message model represents messages on the wire, including some concepts that map closely to domain entities (BookDTO in the sample project is a data clone of Book). In the client application, the View Model (like the BookModel class) is optimized to be bound to the XAML and to handle user interactions.

While at first glance you can see many commonalities among the three models (Book, BookDTO, BookModel), the fact that they have different responsibilities means that trying to cram all of them into a

single model would create a cumbersome, heavyweight, one-size-doesn't-fit-anyone model. By splitting the model along the lines of responsibilities, I made the work much easier because I can refine each model independently to fit its own purposes.

From a conceptual point of view, there are other reasons to want to create a separate model for each usage. An object is a combination of data and behavior, but when you try to send an object over the wire, the only thing you can send is the data. That leads to some interesting questions. Where *do* you place business logic that should run on the back-end server? If you put it in the entities, what happens if you execute this logic on the client?

Figure 1 **Possible Operations on the User's Books Queue**

The end result of this sort of architecture is that you aren't using real objects. Instead, you're using data objects—objects that are simply holding the data—and the business logic resides elsewhere, as procedures that run over the object data. This is frowned upon, because it leads to scattering of logic and code that's harder to maintain over time. No matter how you look at it, unless the back-end system is a simple data repository, you want to have different models in different parts of the application. That, of course, leads to a very interesting question: how are you going to handle changes?

Commands over Change Sets

Among the operations I allow users in the Alexandria application are adding books to their queue, reordering books in the queue, and removing them entirely from the queue, as shown in **Figure 1**. Those operations need to be reflected in both the front end and the back end.

I could try to implement this by serializing the entities over the wire and sending the modified entity back to the server for persistence. Indeed, NHibernate contains explicit support for just such scenarios, using the session.Merge method.

However, let's assume the following business rule: When a user adds a book to her queue from the recommendations list, that book is removed from the recommendations and another recommendation is added.

Imagine trying to detect that a book was moved from the recommendations list to the queue using just the previous and current state (the change set between the two states). While it *can* be done, to say that it would be awkward to handle is an understatement.

I call such architectures Trigger-Oriented Programming. Like triggers in a database, what you have in a system based on change sets is code that deals mostly with data. To provide some meaningful business semantics, you have to extract the meaning of the changes from the change set by brute force and luck.

There's a reason that triggers containing logic are considered an anti-pattern. Though appropriate for some things (such as replication or pure data operations), trying to implement business logic using triggers is a painful process that leads to a system that's hard to maintain.

Most systems that expose a CRUD interface and allow you to write business logic in methods such as UpdateCustomer are giving you Trigger-Oriented Programming as the default (and usually the *only* choice). When there *isn't* significant business logic involved—when the system as a whole is mostly about CRUD—this type of architecture makes sense, but in most applications, it's not appropriate and not recommended.

Instead, an explicit interface (RemoveBookFromQueue and AddBookToQueue, for instance) results in a system that's much easier to understand and think about. The ability to exchange information at this high level allows a great degree of freedom and easy modification down the road. After all, you don't have to figure out where some functionality in the system is based on what data is manipulated by that functionality. The system will spell out exactly where this is happening based on its architecture.

The implementation in Alexandria follows the explicit interface principle; invoking those operations resides in the application model and is shown in **Figure 2**. I'm doing several interesting things here, so let's handle each of these in order.

First, I modify the application model directly to immediately reflect the user's desires. I can do this because adding a book to the user's queue is an operation that is guaranteed never to fail. I also remove it from the recommendations list, because it doesn't make sense to have an item on the user's queue also appear on the recommendations list.

Next, I send a message batch to the back-end server, telling it to add the book to the user's queue, and also to let me know what the user's queue and recommendations are after this change. This is an important concept to understand.

The ability to compose commands and queries in this manner means that you don't take special steps in commands like AddBookToQueue to get the changed data to the user. Instead, the front end can ask for it as part of the same message batch and you can use existing functionality to get this data.

There are two reasons I request the data from the back-end server even though I make the modifications in memory. First, the back-end server may execute additional logic (such as finding new recommendations for this user) that will result in modifications you don't know about on the front-end side. Second, the reply from the back-end server will update the cache with the current status.

Disconnected Local State Management

You might have noticed a problem in **Figure 2** with regard to disconnected work. I make the modification in memory, but

Figure 2 **Adding a Book to the User's Queue on the Front End**

```
public void AddToQueue(BookModel book) {
  Recommendations.Remove(book);
  if (Queue.Any(x => x.Id == book.Id) == false)
    Queue.Add(book);

  bus.Send(
    new AddBookToQueue {
      UserId = userId, BookId = book.Id
    },
    new MyQueueQuery {
      UserId = userId
    },
    new MyRecommendationsQuery {
      UserId = userId
    });
}
```

Figure 3 **Adding a Book to the User's Queue**

```
public class AddBookToQueueConsumer :
  ConsumerOf<AddBookToQueue> {

  private readonly ISession session;

  public AddBookToQueueConsumer(ISession session) {
    this.session = session;
  }

  public void Consume(AddBookToQueue message) {
    var user = session.Get<User>(message.UserId);
    var book = session.Get<Book>(message.BookId);

    Console.WriteLine("Adding {0} to {1}'s queue",
      book.Name, user.Name);

    user.AddToQueue(book);
  }
}
```

until I get a reply back from the server, the cached data isn't going to reflect those changes. If I restart the application while still disconnected, the app will display expired information. Once communication with the back-end server resumes, the messages would flow to the back end and the final state would resolve to what the user is expecting. But until that time, the application is displaying information that the user has already changed locally.

For applications that expect extended periods of disconnection, don't rely only on the message cache; instead implement a model that's persisted after each user operation.

For the Alexandria application, I extended the caching conventions to immediately expire any information that's part of a command-and-queries message batch such as the one in **Figure 2**. That way, I won't have the up-to-date information, but I also won't show erroneous information if the application is restarted before I get a reply from the back-end server. For the purposes of the Alexandria application, that's enough.

Back-End Processing

Now that you understand how the process works on the front-end side of things, let's look at the code from the back-end server point of view. You're already familiar with the handling of queries, which I showed in the previous article. **Figure 3** shows the code for handling a command.

The actual code is pretty boring. I load the relevant entities and then call a method on the entity to perform the actual task. However, this is more important than you might think. An architect's job, I'd argue, is to make sure that the developers in the project are as bored as possible. Most business problems are boring, and by removing technological complexities from the system, you get a much higher percentage of developer time spent working on boring business problems instead of interesting technological problems.

Figure 4 **Removing a Book from the Queue**

```
public void RemoveFromQueue(BookModel book) {
  Queue.Remove(book);

  bus.Send(
    new RemoveBookFromQueue {
      UserId = userId,
      BookId = book.Id
    },
    new MyQueueQuery {
      UserId = userId
    },
    new MyRecommendationsQuery {
      UserId = userId
    });
}
```

What does that mean in the context of Alexandria? Instead of spreading business logic in all the message consumers, I have centralized as much of the business logic as possible in the entities. Ideally, consuming a message follows this pattern:

- Load any data required to process the message
- Call a single method on a domain entity to perform the actual operation

This process ensures that the domain logic is going to remain in the domain. As for what that logic is—well, that's up to the scenarios you need to handle. This should give you an idea about how I handle the domain logic in the case of User.AddToQueue(book):

```
public virtual void AddToQueue(Book book) {
  if (Queue.Contains(book) == false)
    Queue.Add(book);
  Recommendations.Remove(book);

  // Any other business logic related to
  // adding a book to the queue
}
```

You've seen a case where the front-end logic and the back-end logic match exactly. Now let's look at a case where they don't. Removing a book from the queue is very simple on the front end (see **Figure 4**). It's pretty straightforward. You remove the book from the queue locally (which removes it from the UI), then send a message batch to the back end, asking it to remove the book from the queue and update the queue and the recommendations.

On the back end, consuming the RemoveBookFromQueue message follows the pattern shown in **Figure 3**, loading the entities and calling the user.RemoveFromQueue(book) method:

```
public virtual void RemoveFromQueue(Book book) {
  Queue.Remove(book);
  // If it was on the queue, it probably means that the user
  // might want to read it again, so let us recommend it
  Recommendations.Add(book);
  // Business logic related to removing book from queue
}
```

The behavior is different between the front end and the back end. On the back end, I add the removed book to the recommendations, which I don't do on the front end. What would be the result of the disparity?

Well, the immediate response would be to remove the book from the queue, but as soon as the replies from the back-end server reach the front end, you'll see the book added to the recommendations list. In practice, you'd probably be able to notice the difference only if the back-end server was shut down when you remove a book from the queue.

Which is all very nice, but what about when you actually need confirmation from the back-end server to complete an operation?

Complex Operations

When the user wants to add, remove or reorder items in her queue, it's pretty obvious that the operation can never fail, so you can allow the application to immediately accept the operation. But for operations such as editing addresses or changing the credit card, you can't just accept the operation until you have a confirmation of success from the back end.

webdesignworld

Thunder Lizard Productions

SAVE $200 WHEN YOU REGISTER BY SEPTEMBER 15!

LAS VEGAS • OCTOBER 18 - 20, 2010

Three Days of Practical, No-fluff, Top-notch Web Design Instruction

We get it—web design is your world. That's why **Web Design World** is bringing the top minds in web design to **Las Vegas this October**. Join us for three days of the real deal: new sessions devoted to the coding, design, and production techniques you need today, presented by some of the top minds in the field. Oh, and did we mention it's in Vegas?

Here are just a few of the expert speakers you'll find at Web Design World Las Vegas:

Steve Mulder
Vice President,
Experience Strategy Isobar

Greg Rewis
Principal Worldwide Evangelist,
Creative Suite Web Premium,
Adobe Systems, Inc.

Dan Rubin
Designer,
Sidebar Creative

Stephanie Sullivan Rewis
Consultant,
W3Conversions

We take our mantra—**provide practical, no-fluff, how-to**—very seriously. If you've joined us before, you know that. Get the insights you need to build user-friendly sites emphasizing emerging web standards and hands-on learning with the latest tools helping you achieve great design.

Go to WEBDESIGNWORLD.COM today and use Priority Code NQW2 to save $200 when you register before September 15th!

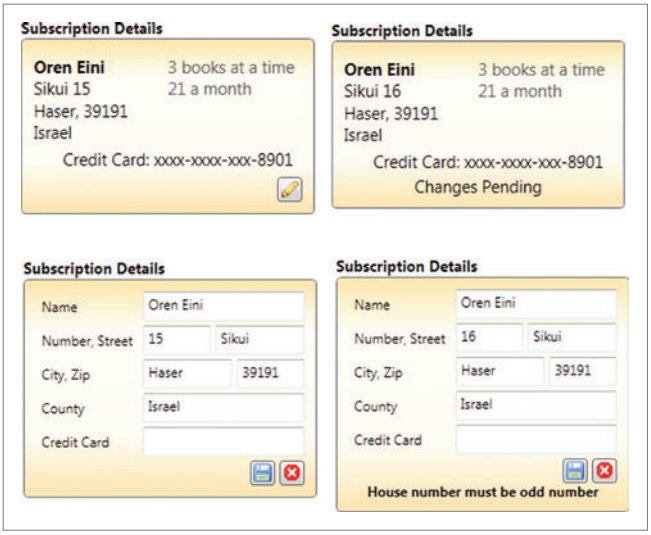


Figure 5 **Four Possible Stages for a Command Requiring Confirmation**

In Alexandria, this is implemented as a four-stage process. It sounds scary, but it's really quite simple. **Figure 5** shows the possible stages.

The top-left screen shot shows the normal view of the subscription details. This is how Alexandria shows confirmed changes. The bottom-left screen shot shows the edit screen for the same data. Clicking the save button on this screen results in the screenshot shown on the top–right; this is how Alexandria shows *unconfirmed* changes.

In other words, I accept the change (provisionally) until I get a reply back from the server indicating that the change was accepted (which moves us back to the top-left screen) or rejected, which moves the process to the bottom-right screenshot. That screenshot shows an error from the server and allows the user to fix the erroneous detail.

The implementation isn't complex, despite what you may think. I'll start in the back end and move outward. **Figure 6** shows the back-end code required to handle this and it isn't anything new. I've been doing much the same thing throughout this article. Most of the conditional command functionality (and complexity) lives in the front end.

One thing that's different from what you've seen before is that here I have explicit success/fail code for the operation, while before I simply requested a data refresh in a separate query. The operation *can* fail, and I want to know not only whether the operation is successful or not, but *why* it failed.

Alexandria makes use of the Caliburn framework to handle much of the drudgery of managing the UI. Caliburn (caliburn.codeplex.com) is a WPF/Silverlight framework that relies heavily on conventions to make it easy to build much of the application functionality in the application model rather than writing code in the XAML code behind.

As you'll see from looking at the sample code, just about everything in the Alexandria UI is wired via the XAML using conventions, giving you both clear and easy to understand XAML and an application model that directly reflects the UI without having a direct dependency on it. This results in significantly simpler code.

Figure 7 should give you an idea about how this is implemented in the SubscriptionDetails view model. In essence, SubscriptionDetails contains two copies of the data; one is held in the Editable property and that's what all the views relating to editing or displaying unconfirmed changes show. The second is held in the Details property, which is used to hold the *confirmed* changes. Each mode has a different view, and each mode selects from which property to display the data.

In the XAML, I wired the ViewMode binding to select the appropriate view to show for every mode. In other words, switching the mode to Editing will result in the Views.SubscriptionDetails.Editing.xaml view being selected to show the edit screen for the object.

It is the save and confirmation processes you will be most interested in, however. Here's how I handle saving:

```
public void Save() {
  ViewMode = ViewMode.ChangesPending;
  // Add logic to handle credit card changes
  bus.Send(new UpdateAddress {
    UserId = userId,
    Details = new AddressDTO {
      Street = Editable.Street,
      HouseNumber = Editable.HouseNumber,
      City = Editable.City,
      ZipCode = Editable.ZipCode,
      Country = Editable.Country,
    }
  });
}
```

The only thing I'm actually doing here is sending a message and switching the view to a non-editable one with a marker saying that those changes have not yet been accepted. **Figure 8** shows the code for confirmation or rejection. All in all, a miniscule amount of code to implement such a feature, and it lays the foundation for implementing similar features in the future.

You also need to consider classic request/response calls, such as searching the catalog. Because communication in such calls is accomplished via one-way messages, you need to change the UI to indicate background processing until the response from the back-end server arrives. I won't go over that process in detail, but the code for doing it exists in the sample application.

Figure 6 **Back-End Handling of Changing a User's Address**

```
public void Consume(UpdateAddress message) {
  int result;
  // pretend we call some address validation service
  if (int.TryParse(message.Details.HouseNumber, out result) ==
    false || result % 2 == 0) {
    bus.Reply(new UpdateDetailsResult {
      Success = false,
      ErrorMessage = "House number must be odd number",
      UserId = message.UserId
    });
  }
  else {
    var user = session.Get<User>(message.UserId);
    user.ChangeAddress(
      message.Details.Street,
      message.Details.HouseNumber,
      message.Details.City,
      message.Details.Country,
      message.Details.ZipCode);

    bus.Reply(new UpdateDetailsResult {
      Success = true,
      UserId = message.UserId
    });
  }
}
```

Figure 7 **Moving Between View Modes in Response to User Input**

```
public void BeginEdit() {
  ViewMode = ViewMode.Editing;

  Editable.Name = Details.Name;
  Editable.Street = Details.Street;
  Editable.HouseNumber = Details.HouseNumber;
  Editable.City = Details.City;
  Editable.ZipCode = Details.ZipCode;
  Editable.Country = Details.Country;
  // This field is explicitly ommitted
  // Editable.CreditCard = Details.CreditCard;
  ErrorMessage = null;
}

public void CancelEdit() {
  ViewMode = ViewMode.Confirmed;
  Editable = new ContactInfo();
  ErrorMessage = null;
}
```

Checking Out

At the beginning of this project, I started by stating the goals and challenges I anticipated facing in building such an application. The major challenges I intended to address were data synchronization, the fallacies of distributed computing, and handling an occasionally connected client. Looking back, I think Alexandria does a good job of meeting my goals and overcoming the challenges.

The front-end application is based on WPF and making heavy use of the Caliburn conventions to reduce the actual code for the application model. The model is bound to the XAML views and a small set of front-end message consumers that make calls to the application model.

I covered handling one-way messaging, caching messages at the infrastructure layer and allowing for disconnected work even for operations that require back-end approval before they can really be considered complete.

Figure 8 **Consuming the Reply and Handling the Result**

```
public class UpdateAddressResultConsumer :
  ConsumerOf<UpdateAddressResult> {
  private readonly ApplicationModel applicationModel;

  public UpdateAddressResultConsumer(
    ApplicationModel applicationModel) {

    this.applicationModel = applicationModel;
  }

  public void Consume(UpdateAddressResult message) {
    if(message.Success) {
      applicationModel.SubscriptionDetails.CompleteEdit();
    }
    else {
      applicationModel.SubscriptionDetails.ErrorEdit(
        message.ErrorMessage);
    }
  }
}

//from SubscriptionDetails
public void CompleteEdit() {
  Details = Editable;
  Editable = new ContactInfo();
  ErrorMessage = null;
  ViewMode = ViewMode.Confirmed;
}

public void ErrorEdit(string theErrorMessage) {
  ViewMode = ViewMode.Error;
  ErrorMessage = theErrorMessage;
}
```

On the back end, I built a message-based application based on Rhino Service Bus and NHibernate. I discussed managing the session and transaction lifetimes and how you can take advantage of the NHibernate first-level cache using messages batches. The message consumers on the back end serve either for simple queries or as delegators to the appropriate method on a domain object, where most of the business logic actually resides.

Forcing the use of explicit commands rather than a simple CRUD interface results in a clearer code. This allows you to change the code easily, because the entire architecture is focused on clearly defining the role of each piece of the application and how it should be built. The end result is a very structured product, with clear lines of responsibility.

It's hard to try to squeeze guidance for a full-blown distributed application architecture into a few short articles, especially while trying to introduce several new concepts at the same time. Still, I think you'll find that applying the practices outlined here will result in applications that are actually easier to work with than the more traditional RPC- or CRUD-based architectures. ■

Oren Eini (who works under the pseudonym Ayende Rahien) is an active member of several open source projects (NHibernate and Castle among them) and is the founder of many others (Rhino Mocks, NHibernate Query Analyzer and Rhino Commons among them). Eini is also responsible for the NHibernate Profiler (nhprof.com), a visual debugger for NHibernate. You can follow his work at ayende.com/Blog.

Windows Phone and the Cloud: an Introduction

Ramon Arjona

I'm learning to read and write in Spanish. Because of where I live, I can practice my Spanish by trying to read the bilingual signs that show up in many public places. The thing is, sometimes I get stuck on one particular word and can't figure it out, or sometimes I'm confused by a whole sentence. It's also tough to be sure that I've understood what I've read without a parallel English translation to refer to. I could carry a Spanish-English dictionary with me, but flipping through all of those pages is just so analog.

What I really want, because I always seem to have my phone in my hand, is an easy way for my phone to help me with the translation. Going to Bing with my phone and using the cloud-based Microsoft Translator is helpful, but it takes a lot of key presses to get to the translation screen. If only there were a way—an easy way—to somehow get the translator on my phone. Well, with the advent of Windows Phone 7, there is.

This article serves as an introduction to developing Windows Phone applications and shows how to tie such an app into a Web service in the cloud. You should have some familiarity with C# and Visual Studio, and it would be helpful if you have at least some experience developing applications with Extensible Application Markup Language (XAML), but this isn't required. I used the April refresh of the community technology preview (CTP) of the Windows Phone tools. By the time you read this, things may have changed, so go to developer.windowsphone.com to read the latest documentation and download the latest tools.

This article uses a community technology preview of the Windows Phone tools. All information is subject to change.

This article discusses:

- Obtaining an AppID to work with Microsoft Translator
- Defining an application user scenario
- Using XAML to design a UI
- Changing an application icon
- Working with the Microsoft Translator API
- Expanding an application with a default list project

Technologies discussed:

Windows Phone 7, Visual Studio Express 2010 for Windows Phone, Microsoft Translator, XAML

First Things First: Getting an AppID

If you're going to interact with Translator—or any of the Bing Web services—the first step is to get an AppID. It's used by the service API to validate that a request is coming from a registered Bing application developer.

Go to bing.com/developers/createapp.aspx and sign in with your Windows Live ID. Then fill out the form, which asks you for your application's name and description, as well as details such as your company name and e-mail address. Your e-mail address is used to notify you when these APIs change, for example, as new versions are rolled out or old versions taken offline. For the Trans-

lator API in particular, at the time of this writing the documentation says that old versions will remain online for 90 days following the release of a new version.

Next, read the terms of use, check the box to signify you accept them and click the "Accept" button, and you'll be taken to a page with your newly minted AppID in it. You can always come back to this page for reference, but I usually copy and paste my AppID into my application's source file when I first create it.

Always read the terms of use for any Web service you plan to consume in an app. Seriously, resist the temptation to just scan them, especially if you plan to distribute your application to users through Marketplace. You owe it to your customers to understand the service-level goals and other conditions set by the partners who provide the services your app consumes. So make sure you read and understand the terms of use, whether your app is consuming a service from Microsoft or from any other party.

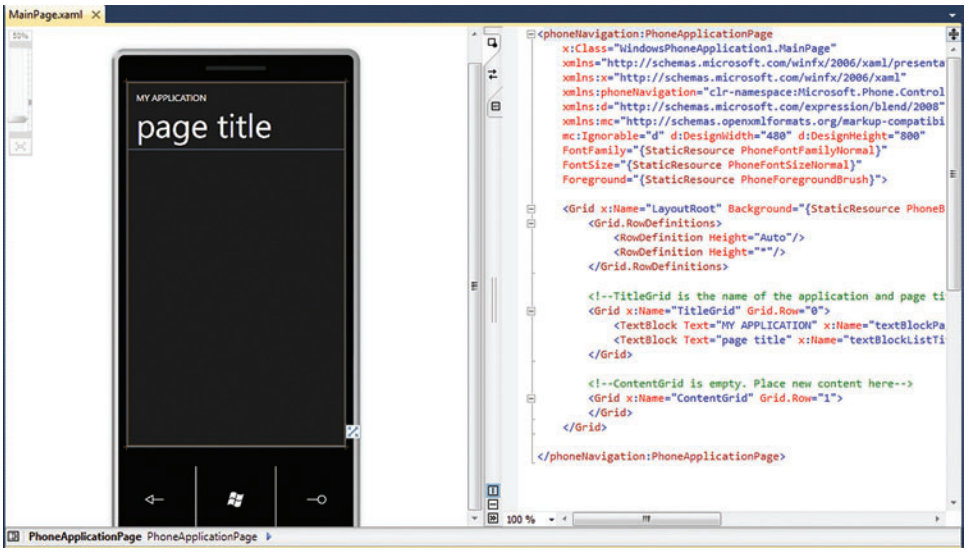


Figure 1 **Default Windows Phone Project with Display and Code Views**

Now that you have an AppID, we're ready to start working on the mobile translator app.

Defining the Scenario

Before I started to work on my first app, I took a look at some successful mobile applications on the market. I looked at applications for Windows phones and other smartphones to try to get a sense of what makes a truly successful mobile application. I read product reviews. I interviewed people. I pestered my friends to let me play with their phones.

After all this research, I concluded that the key to a successful mobile app is simplicity. Every popular app fulfills one or two key user scenarios and fulfills them well. For example, I discovered a whole class of apps whose sole purpose is to generate rude noises. In practice, they're essentially a re-implementation of the whoopee cushion on a phone. People seem to find them amusing, at least enough to pay money to load them onto their phones.

Similarly, there are apps that roll simulated dice, recommend a restaurant for dinner and even calculate the tip amount for that particular dinner. All of these have a focused user scenario and execute on that scenario in a way that's simple and intuitive. This approach isn't unique to any particular brand of smartphone. The idea that we should have a polished, intuitive user experience is reinforced by the Windows Phone UI Design and Interaction Guide published at developer.windowsphone.com/windows-phone-7.

So when I started to work on my translator app, it was important to focus on exactly what I wanted the app to accomplish, and nothing else. To do this, I first had to have a clear picture of what I wanted to accomplish. Once I had this picture in mind, I could resist the temptation to add cool new features that—while interesting—wouldn't serve the main user scenario. I decided my app should translate between English and Spanish. It should easily switch translation modes from Spanish to English or English to Spanish, and it should deal with only single words or simple sentences. Moreover, I only cared about Spanish and English, so my app would ignore other languages.

Building the App

To get started building the translator app, open Visual Studio Express 2010 for Windows Phone. Choose "New project..." and select "Windows Phone Application." This opens the default Windows Phone project with a screen divided into two sections: Title and Content. These sections are delineated with a XAML Grid object, which lets us define a series of rows and columns in which we can position child UI elements such as Buttons and TextBlocks (see **Figure 1**).

First, edit the application name and the page title by selecting these UI elements in the designer. This highlights the related XAML in the code view window of the designer, showing you which element you need to edit. Next, open the toolbox and drag some UI elements to the lower grid on the page, which is named ContentGrid. Choose a TextBox, where we'll put the words we want to translate, then a TextBlock, where the translation will be rendered. Finally, add two buttons (see **Figure 2**).

One button will execute the translation and the second will toggle the direction of the translation from Spanish to English or back again.

XAML: the Modern UI

The heart of the Windows Phone 7 user experience is the design system code-named "Metro." And the heart of implementing a UI according to the Metro system is XAML. If you've worked with Silverlight or Windows Presentation Foundation (WPF) before, XAML is probably familiar to you. If you haven't, this section provides a brief introduction.

XAML is a way to declaratively structure your UI, giving you a strict separation between your app's logic and its look and feel. You'll notice that the code for your app's main page, for example, is

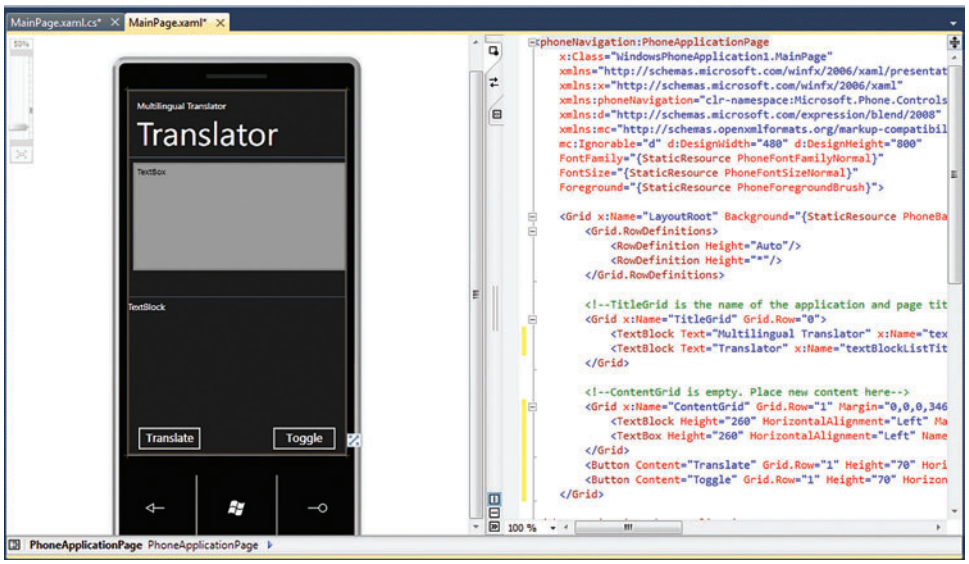


Figure 2 **Adding TextBox, TextBlock and Button UI Elements**

contained in Mainpage.xaml.cs, but the basic layout of your app is contained in Mainpage.xaml. Unlike traditional Windows Forms development, there's no mixing of app code with app UI layout. Every UI element in your application is modeled as an XML element in a XAML file.

This is an advantage because it allows one person to work independently on the app's look and feel while another works on the app's logic. For example, now that we've outlined the basic UI of our translator app, we could hand the XAML to a UI expert while we continue to work on the code. The designer wouldn't need to know anything about the implementation of my app and I wouldn't need to know anything about the color palette he's chosen.

Of course, I don't have any budget for a UI designer, but I do have MSPaint, and that's enough to demonstrate how easy it is to update the graphic assets that accompany my app.

Giving the Translator a New Icon

I double-click the default ApplicationIcon.png that was automatically added to the project when I created it. This opens my image-editing software, which is MSPaint on my dev machine. I replace the boring gear icon with a capital letter T. Then I erase the black background and fill it in with a monochrome pink. I probably won't win any awards for graphic design, but I like it (see **Figure 3**).

To review the change, I hit the start button to deploy the application to the Windows Phone 7 emulator. If you've worked with previous versions of Windows Phone emulators, you'll see right away how much the developer experience has improved. Working with the emulator for Windows Phone 6.5 can be a pain and involves manual configuration steps to start and attach the debugger to the emulator. Here in the Windows Phone 7 development environment, everything just works once I hit the green start button. I'm even happy with how quickly the emulator loads and displays the app's main UI (see **Figure 4**).

To see the app icon, I exit from the main UI and navigate to the emulator's main screen. The translator graphic is there, and it looks OK (see **Figure 5**).

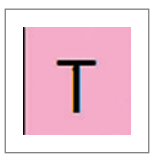


Figure 3 **A New Project Icon Created with MSPaint**

The App Manifest

The application icon is defined in a file called WMAppManifest.xml, which is located in your project's Properties folder. To rename your app icon, change the name of the ApplicationIcon.png in your project and then make sure you reflect the change here, in the IconPath element:

```
<IconPath IsRelative="true"
 IsResource="false">myicon.png
 </IconPath>
```

If your app can't find its icon, it will be assigned the runtime default, which looks something like a white circle on a black background.

Another element of this file that bears examination is the Capabilities element, which contains the specific capabilities your application wants to use. It's a good idea to request exactly what your application needs, nothing more. If you scan the file, you'll see it's got a number of capabilities we probably aren't going to use in our translator application.

For example, ID_CAP_GAMERSERVICES declares that your app needs to interact with the XBox game APIs. The ID_CAP_LOCATION capability says that the app wants to make use of the device's location capability. And ID_CAP_PUSH_NOTIFICATION says that the app wants to interact with the push notification feature of Windows Phone 7. All of these are great features, but we don't need them for our app, so out they go. In fact, we probably need only the ID_CAP_NETWORKING capability, which says our app wants to use the network to send and receive data.

The Microsoft Translator API

Microsoft Translator exposes three kinds of APIs. The SOAP API gives the consumer strong typing and ease of use. The AJAX API is primarily useful for Web page developers who want to embed translation into their UI. The HTTP API is useful when neither the SOAP nor the AJAX API is appropriate.

We're going to choose the SOAP interface, because it's easiest to work with for our purposes. Go to Solution Explorer and right-click References. Then choose "Add a Service Reference" and input the endpoint for the SOAP interface to the Translator API: http://api.microsofttranslator.com/V2/Soap.svc. Give the service endpoint the name TranslatorService in the namespace textbox and click OK (see **Figure 6**).

Visual Studio takes care of the rest, generating the SOAP interface client code for you.

Now add some code to your application. Add the AppID and a reference to the TranslatorService client in the declaration of your app's MainPage class:

```
string appID = <<your appID>>;
TranslationService.LanguageServiceClient client =
  new TranslationService.LanguageServiceClient();
```

IntelliSense shows us that the translator service has a number of interesting methods available. The first thing to notice is that all of the methods are asynchronous. This makes

Figure 4 **The App Displayed by the Windows Phone 7 Emulator**

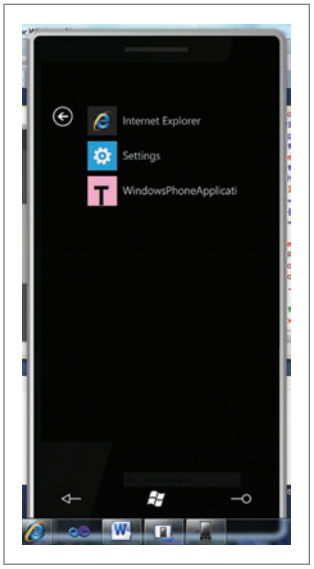


Figure 5 **Checking the New App Icon in the Emulator**

sense, because there's no good reason to block the client application while we're waiting for a network operation to complete. It means we'll need to register a delegate for each specific operation we perform. In our case, we're only interested in the TranslateAsync method and the TranslateAsyncComplete event exposed by the LanguageServiceClient. But the other methods certainly do look interesting. Let's take a look at them and see if they fit into our user scenario.

Two methods that stand out are GetLanguagesForTranslateAsync and GetLanguageNamesAsync. The first method provides a list of language codes supported by the Translator service. By calling this method, you can see the language code for Spanish is "es." The GetLanguageNamesAsync method returns a list of language names for a given set of language codes, localized for a given locale. For example, if you pass "es" into this method as the locale and the language code, you get back the string "Español." If we were doing a multilingual translator, these would both be useful.

Another interesting method is named SpeakAsync. It accepts a string and a locale and returns a URL to a WAV file that's the native-sounding pronunciation of a word. This is an awesome feature. For example, I could type in a string, get the translation and then pass this string to the SpeakAsync method to get a WAV file I could use to communicate with others in Spanish. Or if I were unsure of the pronunciation of a particular word, I could use the SpeakAsync method to hear what it was supposed to sound like.

These are pretty cool, and it's tough to resist the temptation to put them into the app just because they're cool. For now, though, we should stand firm and focus only on fulfilling the user scenario we outlined at the start. Having this clear picture in mind makes it easier when faced with the temptation to add "just one more feature." I'm sure I'll end up using the features offered in the Translator API at some point, just not right now.

Wiring up the code to make our translator app is easy. First, we register the delegates for TranslateCompleted:

```
client.TranslateCompleted += new
  EventHandler<TranslationService.TranslateCompletedEventArgs>

(client_TranslateCompleted);
```

Then we implement the event handler for the TranslateCompleted, which sets the text of our TextBlock to the translated text:

```
void client_TranslateCompleted(object sender,
  TranslationService.TranslateCompletedEventArgs e)
{
  TranslatedTextBlock.Text = e.Result;
}
```

We wire up the button to submit the text we've entered for translation:

```
private void TranslateButton_Click(object sender,
  RoutedEventArgs e)
{
  client.TranslateAsync(appID, TranslateTextBox.Text, fromLanguage,
    toLanguage);
}
```

Then we add some simple code to the second button to toggle between translation modes, from "Spanish to English" or from "English to Spanish." This button manages a global state variable and changes the text of our translate button to indicate that state.

Finally, we deploy the app to the emulator to test it. We've now got a fully functional translator app, with just a few lines of code and an hour or so of development time.

Expanding the Translator Application

The simple translator app is nice, but we could take it further by expanding our user scenario. It would be great if we could tie more of the languages offered by the Translator API into our app in a way that fits with the Metro design guidelines. And it would be even better if we could eliminate the button that manages the translation direction and instead have a simpler, more intuitive way of managing the direction of the translation. Whatever we do, it has to be friendly to a person using his fingers to interact with the device, and it has to feel fast and easy.

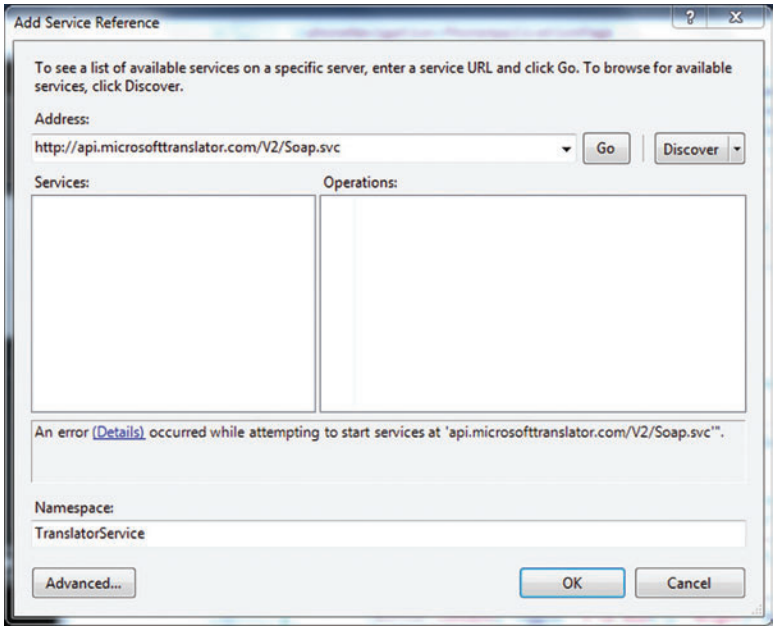


Figure 6 **Adding a SOAP Service Reference**

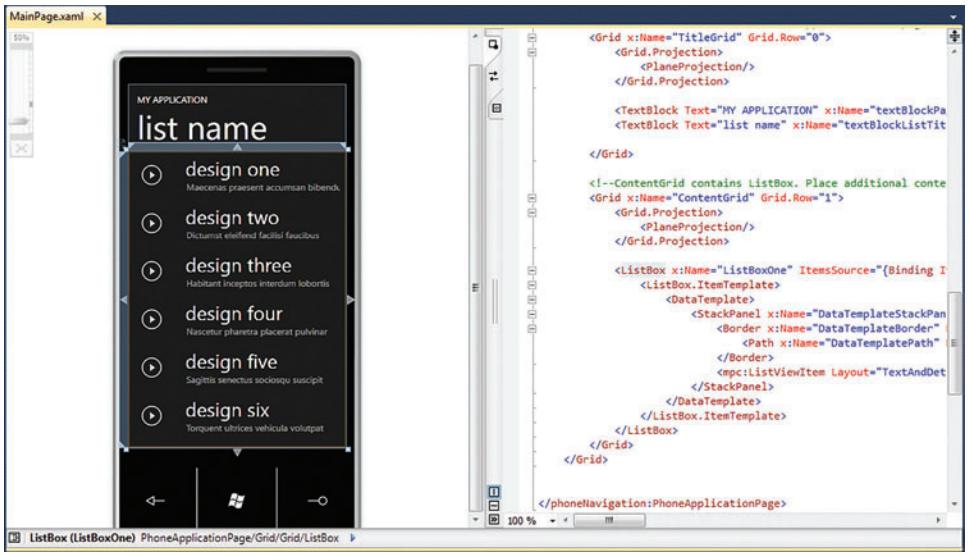


Figure 7 **The Default Windows Phone List Application**

So what do we do? We build an application using the default list project. Start a new project and select "Windows Phone List Application." This gives you a default screen with a ListBox control in the ContentGrid. The ListBox will contain six languages we want to translate our text into, replacing the dummy text (see **Figure 7**).

We'll edit the MainViewModelSampleData.xaml file, which is contained in the SampleData folder in your project. You'll see some XML that looks something like this:

```
<local:ItemViewModel LineOne="design one" LineTwo="Maecenas praesent
accumsan bibendum" LineThree="Maecenas praesent accumsan bibendum dictumst
eleifend facilisi faucibus habitant inceptos interdum lobortis nascetur"/>
```

By editing the XML, you can change the design-time view of the ListBox to include all of the languages we want to work with: Spanish, German, English, Portuguese, Italian and French. Notice that after you save this file, the design-time view will reflect your changes. This is because the ListBox control is databound to the Items element defined in MainViewModelSampleData.xaml. This is controlled by the ItemsSource attribute of the ListBox control.

Figure 8 **The List Application UI**

You can also update the contents of the ListBox at run time. If you look into the file generated by Visual Studio, you'll see an ObservableCollection<ItemViewCollection> being filled with placeholder data similar to that contained in the MainViewModelSampleData.xaml file. For example, if I wanted to dynamically generate the list of languages in the UI based on the languages returned by the GetLanguageNamesAsync method, I would edit the MainViewModel.cs file to populate the Items collection. I prefer to specify a static list of languages for now and just edit the collection of ItemViewModel objects directly. Now when we run the list application, we should see a UI something like **Figure 8**.

When the user touches one of these buttons, he'll be taken to a details page that looks essentially the same as the UI of our SimpleTranslator. There's a TextBox to enter the words to translate, a TextBlock to contain the translated text and a button to submit the translation. Instead of adding another button to manage the direction of the translation, however, we'll rely on the Translator API. The bidirectional translation of our SimpleTranslator app doesn't make sense in this multilingual scenario, and adding additional layers of UI would make the app start to feel clunky instead of smooth. Fortunately, the Translator API provides a way to automatically detect the language being passed in.

We add some code to first call the DetectAsync method of the translator client, passing in the text we want to translate, then call TranslateAsync from the DetectAsyncComplete event handler. The event handler now looks like this:

```
void client_DetectCompleted(object sender,
  TranslationService.DetectCompletedEventArgs e)
{
  string languageCode = e.Result;
  client.TranslateAsync(appID, TranslateTextBox.Text, fromLanguage, tolanguage);
}
```

We know the language we want to translate into because of the button the user selected. We know the language that we want to translate from because the Translator API has autodetected it. We're able to create a simple, multilingual translator that allows the user to get a translation with just two touches—not counting typing—and less than an hour of coding.

Simplicity

Simplicity is at the heart of development for Windows Phone 7. An app should focus on fulfilling one user scenario, and fulfilling it well. The tools that come with the Windows Phone 7 SDK CTP make it not only possible but easy to deliver apps that are both simple and powerful. We've also seen that interacting with the cloud is straightforward, and that the Microsoft Translator Web service can be wired into a Windows Phone app with just a small investment of developer time. By taking familiar tools such as Visual Studio and clear guidelines such as Metro, Windows Phone and the cloud come together to open up a world of new opportunities for developers and users. ■

RAMON ARJONA *is a senior test lead working on the Windows team at Microsoft.*

THANKS to the following technical experts for reviewing this article: *VikramDendi and Sandor Maurice*

NOVEMBER 14–17, 2010
ORLANDO, FL | HILTON WALT DISNEY WORLD RESORT

1105 MEDIA

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REAL-WORLD TRAINING. KILLER SKILLS.

Register today for Visual Studio Live! Orlando and learn how to maximize the development capabilities of Visual Studio, .NET 4.0, and so much more. Join us for four action-packed days of workshops, 50+ sessions by expert instructors, and keynotes by industry heavyweights. Topics covered include:

- PROGRAMING WITH WCF
- ARCHITECTING FOR AZURE
- WPF & SILVERLIGHT
- ASP.NET 4
- jQUERY FOR ASP.NET
- WHAT'S NEW IN VISUAL STUDIO 2010
- SHAREPOINT 2010 FOR ASP.NET DEVELOPERS

DETAILS AND REGISTRATION AT

VSLIVE.COM/ORLANDO

USE PRIORITY CODE NQZF1

Supported by: Microsoft msdn Microsoft Visual Studio Visual Studio MAGAZINE

JAMES MCCAFFREY

Fault Injection Testing with TestApi

Fault injection testing is the process of deliberately inserting an error into an application under test and then running the application to determine whether the application deals with the error properly. Fault injection testing can take several different forms. In this month's column, I explain how you can introduce faults into .NET applications at run time using a component of the TestApi library.

The best way for you to see where I'm headed in this column is to take a look at the screenshot in **Figure 1**. The screenshot shows that I'm performing fault injection testing on a dummy .NET WinForm application named TwoCardPokerGame.exe. A C# program named FaultHarness.exe is running in the command shell. It alters the normal behavior of the application under test so the application throws an exception the third time a user clicks on the button labeled Evaluate. In this situation, the Two Card Poker application does not handle the application exception gracefully and the result is the system-generated message box.

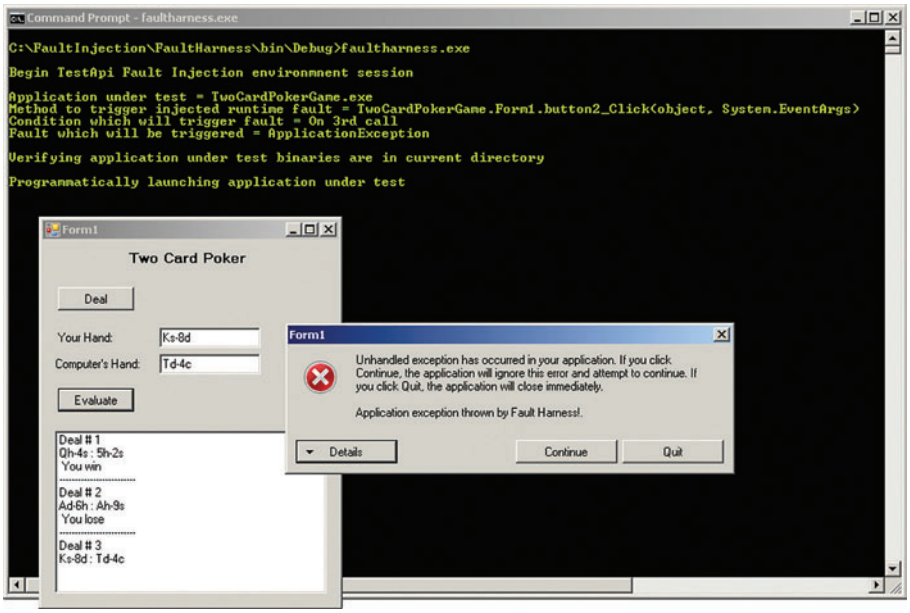


Figure 1 **Fault Injection Testing in Action**

Let's take a closer look at this scenario to consider some of the details involved. When FaultHarness.exe is launched from the command shell, behind the scenes the harness prepares profiling code that will intercept the normal code execution of TwoCardPokerGame.exe. This is called the fault injection session.

The fault injection session uses a DLL to start watching for calls to the application's button2_Click method, which is the event handler for the button labeled Evaluate. The fault injection session has been configured so that the first two times a user clicks on the Evaluate button, the application behaves as coded, but on the third click the fault session causes the application to throw an exception of type System.ApplicationException.

The fault session records session activity and logs a set of files to the test host machine. Notice in **Figure 1** that the first two application Deal-Evaluate click pairs work properly, but the third click generated an exception.

In the sections that follow, I'll briefly describe the dummy Two Card Poker Game application under test, present and explain in detail the code in the FaultHarness.exe program shown in **Figure 1**, and provide some tips about when the use of fault injection testing is appropriate and when alternative techniques are more suitable. Although the FaultHarness.exe program itself is quite simple and most of the difficult work is performed behind the scenes by the TestApi DLLs, understanding and modifying the code I present here to meet your own testing scenarios requires a solid understanding of the .NET programming environment. That said, even if you're a .NET beginner, you should be able to follow my explanations without too much difficulty. I'm confident you'll find the discussion of fault injection an interesting and possibly useful addition to your toolset.

The Application Under Test

My dummy application under test is a simplistic but representative C# WinForm application that simulates a hypothetical card game called Two Card Poker. The application consists of two main components: TwoCardPokerGame.exe provides the UI and TwoCardPokerLib.dll provides the underlying functionality.

To create the game DLL I launched Visual Studio 2008 and selected the C# Class Library template from the File | New Project

Code download available at code.msdn.microsoft.com/mag201008TestRun.

dialog box. I named the library TwoCardPokerLib. The overall structure of the library is presented in **Figure 2**. The code for TwoCardPokerLib is too long to present in its entirety in this article. The complete source code for the TwoCardPokerLib library and the FaultHarness fault injection harness is available in the code download that accompanies this article.

The Application UI Code

Once I had the underlying TwoCardPokerLib library code finished, I created a dummy UI component. I started a new project in Visual Studio 2008 using the C# WinForm Application template and I named my application TwoCardPokerGame.

Figure 2 **The TwoCardPokerLib Library**

```
using System;
namespace TwoCardPokerLib {
  // ------------------------------------------------
  public class Card {
    private string rank;
    private string suit;
    public Card() {
      this.rank = "A"; // A, 2, 3, . . ,9, T, J, Q, K
      this.suit = "c"; // c, d, h, s
    }
    public Card(string c) { . . . }
    public Card(int c) { . . . }
    public override string ToString(){ . . . }
    public string Rank { . . . }
    public string Suit { . . . }
    public static bool Beats(Card c1, Card c2) { . . . }
    public static bool Ties(Card c1, Card c2) { . . . }
  } // class Card

  // ------------------------------------------------
  public class Deck {
    private Card[] cards;
    private int top;
    private Random random = null;

    public Deck() {
      this.cards = new Card[52];
      for (int i = 0; i < 52; ++i)
        this.cards[i] = new Card(i);
      this.top = 0;
      random = new Random(0);
    }

    public void Shuffle(){ . . . }
    public int Count(){ . . . }
    public override string ToString(){ . . . }
    public Card[] Deal(int n) { . . . }

  } // Deck

  // ------------------------------------------------
  public class Hand {
    private Card card1; // high card
    private Card card2; // low card
    public Hand(){ . . . }
    public Hand(Card c1, Card c2) { . . . }
    public Hand(string s1, string s2) { . . . }
    public override string ToString(){ . . . }
    private bool IsPair() { . . . }
    private bool IsFlush() { . . . }
    private bool IsStraight() { . . . }
    private bool IsStraightFlush(){ . . . }
    private bool Beats(Hand h) { . . . }
    private bool Ties(Hand h) { . . . }
    public int Compare(Hand h) { . . . }
    public enum HandType { . . . }

  } // class Hand

} // ns TwoCardPokerLib
```

Using the Visual Studio designer, I dragged a Label control from the Toolbox collection onto the application design surface, and modified the control's Text property from "textBox1" to "Two Card Poker." Next I added two more Label controls ("Your Hand" and "Computer's Hand"), two TextBox controls, two Button controls ("Deal" and "Evaluate"), and a ListBox control. I didn't change the default control names of any of the eight controls—textBox1, textBox2, button1 and so on.

Once my design was in place, I double-clicked on the button1 control to have Visual Studio generate an event handler skeleton for the button and load file Form1.cs into the code editor. At this point I right-clicked on the TwoCardPokerGame project in the Solution Explorer window, selected the Add Reference option from the context menu, and pointed to the file TwoCardPokerLib.dll. In Form1.cs, I added a using statement so that I wouldn't need to fully qualify the class names in the library.

Next, I added four class-scope static objects to my application:

```
namespace TwoCardPokerGame {
  public partial class Form1 : Form {
    static Deck deck;
    static Hand h1;
    static Hand h2;
    static int dealNumber;
...
```

Object h1 is the Hand for the user, and h2 is the Hand for the computer. Then I added some initialization code to the Form constructor:

```
public Form1() {
  InitializeComponent();
  deck = new Deck();
  deck.Shuffle();
  dealNumber = 0;
}
```

The Deck constructor creates a deck of 52 cards, in order from the ace of clubs to the king of spades, and the Shuffle method randomizes the order of the cards in the deck.

Next I added the code logic to the button1_Click method as shown in **Figure 3**. For each of the two hands, I call the Deck.Deal method to remove two cards from the deck object. Then I pass those two cards to the Hand constructor and display the value of the hand in a TextBox control. Notice that the button1_Click method handles any exception by displaying a message in the ListBox control.

Next, in the Visual Studio designer window I double-clicked on the button2 control to auto-generate the control's event handler skeleton. I added some simple code to compare the two Hand objects and display a message in the ListBox control. Notice that the button2_Click method does not directly handle any exceptions:

```
private void button2_Click(
  object sender, EventArgs e) {
  int compResult = h1.Compare(h2);
  if (compResult == -1)
    listBox1.Items.Add(" You lose");
  else if (compResult == +1)
    listBox1.Items.Add(" You win");
  else if (compResult == 0)
    listBox1.Items.Add(" You tie");

  listBox1.Items.Add("-------------------------");
}
```

The Fault Injection Harness

Before creating the fault injection harness shown in **Figure 1**, I downloaded the key DLLs to my test host machine. These DLLs are part of a collection of .NET libraries named TestApi and can be found at testapi.codeplex.com.

Figure 3 **Dealing the Cards**

```
private void button1_Click(
  object sender, EventArgs e) {

  try  {
    ++dealNumber;
    listBox1.Items.Add("Deal # " + dealNumber);
    Card[] firstPairOfCards = deck.Deal(2);
    h1 = new Hand(firstPairOfCards[0], firstPairOfCards[1]);
    textBox1.Text = h1.ToString();

    Card[] secondPairOfCards = deck.Deal(2);
    h2 = new Hand(secondPairOfCards[0], secondPairOfCards[1]);
    textBox2.Text = h2.ToString();
    listBox1.Items.Add(textBox1.Text + " : " + textBox2.Text);
  }
  catch (Exception ex) {
    listBox1.Items.Add(ex.Message);
  }
}
```

The TestApi library is a collection of software-testing-related utilities. Included in the TestApi library is a set of Managed Code Fault Injection APIs. (Read more about them at blogs.msdn.com/b/ivo_manolov/archive/2009/11/25/9928447.aspx.) I downloaded the latest fault injection APIs release, which in my case was version 0.4, and unzipped the download. I will explain what's in the download and where to place the fault injection binaries shortly.

Version 0.4 supports fault injection testing for applications created using the .NET Framework 3.5. The TestApi library is under active development, so you should check the CodePlex site for updates to the techniques I present in this article. Additionally, you may want to check for updates and tips on the blog of Bill Liu, the primary developer of the TestApi fault injection library, at blogs.msdn.com/b/billliu/.

To create the fault injection harness I started a new project in Visual Studio 2008 and selected the C# Console Application template. I named the application FaultHarness and I added some minimal code to the program template (see **Figure 4**).

I hit the <F5> key to build and run the harness skeleton, which created a \bin\Debug folder in the FaultHarness root folder.

The TestApi download has two key components. The first is TestApiCore.dll, which was located in the Binaries folder of the unzipped download. I copied this DLL into the root directory of the FaultHarness application. Then I right-clicked on the FaultHarness project in the Solution Explorer window, selected Add Reference,

Figure 4 **FaultHarness**

```
using System;
namespace FaultHarness {
  class Program {
    static void Main(string[] args) {
      try {
        Console.WriteLine("\nBegin TestApi Fault Injection environmnent
session\n");

        // create fault session, launch application

        Console.WriteLine("\nEnd TestApi Fault Injection environment session");
      }
      catch (Exception ex) {
        Console.WriteLine("Fatal: " + ex.Message);
      }
    }
  } // class Program
} // ns
```

and pointed it to TestApiCore.dll. Next, I added a using statement for Microsoft.Test.FaultInjection to the top of my fault harness code so my harness code could directly access the functionality in TestApiCore.dll. I also added a using statement for System.Diagnostics because, as you'll see shortly, I want to access the Process and ProcessStartInfo classes from that namespace.

The second key component in the fault injection download is a folder named FaultInjectionEngine. This holds 32-bit and 64-bit versions of FaultInjectionEngine.dll. I copied the entire FaultInjectionEngine folder into the folder holding my FaultHarness executable, in my case C:\FaultInjection\FaultHarness\bin\Debug\. The 0.4 version of the fault injection system I was using requires the FaultInjectionEngine folder to be in the same location as the harness executable. Additionally, the system requires that the application under test binaries be located in the same folder as the harness executable, so I copied files TwoCardPokerGame.exe and TwoCardPokerLib.dll into C:\FaultInjection\FaultHarness\bin\Debug\.

To summarize, when using the TestApi fault injection system, a good approach is to generate a skeleton harness and run it so that a harness \bin\Debug directory is created, then place file TestApiCore.dll in the harness root directory, place the FaultInjectionEngine folder in \bin\Debug, and place the application under test binaries (.exe and .dll) in \bin\Debug as well.

Using the TestApi fault injection system requires that you specify the application under test, the method in the application under test that will trigger a fault, the condition that will trigger a fault, and the kind of fault that will be triggered:

```
string appUnderTest = "TwoCardPokerGame.exe";
string method =
  "TwoCardPokerGame.Form1.button2_Click(object, System.EventArgs)";
ICondition condition =
  BuiltInConditions.TriggerEveryOnNthCall(3);
IFault fault =
  BuiltInFaults.ThrowExceptionFault(
    new ApplicationException(
    "Application exception thrown by Fault Harness!"));
FaultRule rule = new FaultRule(method, condition, fault);
```

Notice that, because the system requires the application under test to be in the same folder as the harness executable, the name of the application under test executable does not need the path to its location.

Specifying the name of the method that will trigger the injected fault is a common source of trouble for TestApi fault injection beginners. The method name must be fully qualified in the form Namespace.Class.Method(args). My preferred technique is to use the ildasm.exe tool to examine the application under test to help me determine the triggering method's signature. From the special Visual Studio tools command shell I launch ildasm.exe, point to the application under test, then double-click on the target method. **Figure 5** shows an example of using ildasm.exe to examine the signature for the button2_Click method.

When specifying the trigger method signature, you do not use the method return type, and you do not use parameter names. Getting the method signature correct sometimes requires a bit of trial and error. For example, on my first attempt to target button2_Click, I used:

```
TwoCardPokerGame.Form1.button2_Click(object,EventArgs)
```

I had to correct it to:

```
TwoCardPokerGame.Form1.button2_Click(object,System.EventArgs)
```

The TestApi download contains a Documentation folder containing a concepts document that provides good guidance on how to

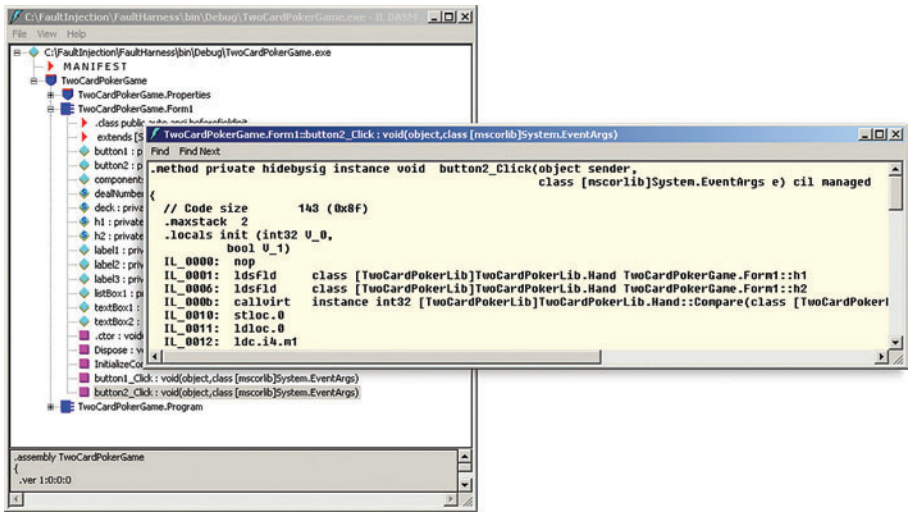


Figure 5 **Using ILDASM to Examine Method Signatures**

When you execute the fault harness, it will launch the application under test in your fault session, with the FaultInjectionEngine.dll watching for situations where the trigger method is called when the trigger condition is true. The tests are performed manually here, but you can also run test automation in a fault session.

While the fault session is running, information about the session is logged into the current directory—that is, the directory that holds the fault harness executable and the application under test executable. You can examine these log files to help resolve any problems that might occur while you're developing your fault injection harness.

correctly construct different kinds of method signatures including constructors, generic methods, properties, and overloaded operators. Here I target a method that's located in the application under test, but I could have also targeted a method in the underlying TwoCardPokerLib.dll, such as:

```
string method = "TwoCardPokerLib.Deck.Deal(int)"
```

After specifying the trigger method, the next step is to specify the condition under which the fault will be injected into the application under test. In my example I used TriggerEveryOnNthCall(3), which as you've seen injects a fault every third time the trigger method is called. The TestApi fault injection system has a neat set of trigger conditions including TriggerIfCalledBy(method), TriggerOnEveryCall, and others.

After specifying the trigger condition, the next step is to specify the type of fault that will be injected into the system under test. I used BuiltInFaults.ThrowExceptionFault. In addition to exception faults, the TestApi fault injection system has built-in return type faults that allow you to inject erroneous return values into your application under test at run time. For example, this will cause the trigger method to return a (presumably incorrect) value of -1:

```
IFault f = BuiltInFaults.ReturnValueFault(-1)
```

After the fault trigger method, condition, and fault kind have been specified, the next step is to create a new FaultRule and pass that rule to a new FaultSession:

```
FaultRule rule = new FaultRule(method, condition, fault);
Console.WriteLine(
  "Application under test = " + appUnderTest);
Console.WriteLine(
  "Method to trigger injected runtime fault = " + method);
Console.WriteLine(
  "Condition which will trigger fault = On 3rd call");
Console.WriteLine(
  "Fault which will be triggered = ApplicationException");
FaultSession session = new FaultSession(rule);
```

With all the preliminaries in place, the last part of writing the fault harness code is to programmatically launch the application under test in the fault session environment:

```
ProcessStartInfo psi =
  session.GetProcessStartInfo(appUnderTest);
Console.WriteLine(
  "\nProgrammatically launching application under test");
Process p = Process.Start(psi);
p.WaitForExit();
p.Close();
```

Discussion

The example and explanations I've presented here should get you up and running with creating a fault injection harness for your own application under test. As with any activity that's part of the software development process, you will have limited resources and you should analyze the costs and benefits of performing fault injection testing. In the case of some applications, the effort required to create fault injection testing may not be worthwhile, but there are many testing scenarios where fault injection testing is critically important. Imagine software that controls a medical device or a flight system. In situations such as these, applications absolutely must be robust and able to correctly handle all kinds of unexpected faults.

There is a certain irony involved with fault injection testing. The idea is that, if you can anticipate the situations when an exception can occur, you can in theory often programmatically guard against that exception and test for the correct behavior of that guarding behavior. However, even in such situations, fault injection testing is useful for generating difficult to create exceptions. Additionally, it's possible to inject faults that are very difficult to anticipate, such as System.OutOfMemoryException.

Fault injection testing is related to and sometimes confused with mutation testing. In mutation testing, you deliberately insert errors into the system under test, but then execute an existing test suite against the faulty system in order to determine whether the test suite catches the new errors created. Mutation testing is a way to measure test suite effectiveness and ultimately increase test case coverage. As you've seen in this article, the primary purpose of fault injection testing is to determine whether the system under test correctly handles errors. ■

Dr. James McCaffrey *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He's worked on several Microsoft products, including Internet Explorer and MSN Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006) and can be reached at jammc@microsoft.com.*

Thanks to the following technical experts for reviewing this article: *Bill Liu and Paul Newson*

TED NEWARD

Inside SQLite

In keeping with the theme of this issue, it's time to return to the roots of SQL and relational databases. Naturally, it would thus seem apropos to write something about SQL Server, something about its new feature set or performance improvements or whatnot, but that's just not my style. Don't get me wrong, SQL Server is a great database, and highly recommended for those "big iron" enterprise-class scenarios, but not every problem demands (as a friend once put it) a "big honkin' centralized database."

In fact, developers have long been using relational databases merely as a place to "put stuff for next time"—stuff such as configuration options, user settings, internationalization values and so on. Although it's sometimes convenient to put these into a centralized SQL Server instance, in some cases, particularly in rich client scenarios (and especially Microsoft Silverlight or Windows Phone 7 rich client scenarios), maintaining a constant connection back to the SQL Server instance is infeasible at best, and often just flat-out impossible.

Developers don't necessarily want to give up the power and flexibility of a relational database, but even SQL Server Express is sometimes too heavy an install. What's to be done?

Go light, of course: SQLite, to be precise.

Introducing SQLite

As described by its Web site (sqlite.org), "SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine." The key elements in that statement revolve around the noun "library." Unlike SQL Server, which uses a client-side assembly to send requests to a server for parsing and execution, SQLite lives entirely inside the client process, making it an "embedded" database. The running footprint of a SQLite database during use is a single file stored someplace on the client file system, and typically the install footprint is equally small.

For all that, the SQLite database is remarkably feature-rich, in that it supports the majority of the SQL-92 specification, minus a few things (described in more detail on the SQLite Web site) such as RIGHT and FULL OUTER JOIN, ALTER TABLE, some trigger support, GRANT/REVOKE and writing to VIEWs. What's impressive is how much *is* supported, including transactions and a wide range of data types. Although it's probably beyond credibility to expect that a SQL Server database schema will port to SQLite without modification, it's reasonable to assume that a fairly straightforward (that is, not taking advantage of SQL Server-specific types or features) schema will port with minimal trouble. This makes SQLite ideal for scenarios where just a "lightweight SQL" is necessary.

In case there are concerns about its applicability or stability, SQLite is slowly making its way into a variety of "lightweight" environments—it already appears inside the Mozilla Firefox browser (for HTML 5 support), as well as the Symbian, iOS and Android environments, among others. In other words, this is how the "other half" of the development world (that is, non-Microsoft-centric) does the lightweight database. SQLite continues to enjoy ongoing development and bug fixing, making it a pretty safe bet as a minimal SQL engine.

> Not every problem demands (as a friend once put it) a "big honkin' centralized database."

Of course, no database would be complete without some kind of administrator interface, and SQLite doesn't disappoint. It has a command-line console tool for accessing and manipulating SQLite databases—but that may not impress your sysadmin all that much. Fortunately, the open source community has a number of SQLite tools available (a long list of them is on the SQLite Web site), but if you just need a quick Query Analyzer-like tool, try SQLite Administrator, a free tool available for download at sqliteadmin.orbmu2k.de.

Goin' Native

SQLite was intended from the beginning to be a database for the native code developer, which is why it's implemented as a native C/C++ DLL. This native flavor of SQLite is both a blessing and a curse: blessing, in that it cuts out a lot of the overhead (such as that of traversing the network to the server and back again) from the total time required to execute a given SQL statement; but curse in that, because the original SQLite database is a native C/C++ DLL, getting to it from a Microsoft .NET Framework-based application can be challenging.

Fortunately, the savvy .NET Framework developer realizes that accessing a native DLL is really just an exercise in P/Invoke declarations, and it's relatively easy to create a wrapper class around the native declarations exposed in the SQLite DLL. In fact, for the

Code download available at code.msdn.microsoft.com/mag201008WorkProg.

Figure 1 **P/Invoke Declarations**

```
namespace SQLiteWrapper
{
  public class SQLiteException : Exception
  {
    public SQLiteException(string message) :
      base(message)
      { }
  }

  public class SQLite
  {
    const int SQLITE_OK = 0;
    const int SQLITE_ROW = 100;
    const int SQLITE_DONE = 101;
    const int SQLITE_INTEGER = 1;
    const int SQLITE_FLOAT = 2;
    const int SQLITE_TEXT = 3;
    const int SQLITE_BLOB = 4;
    const int SQLITE_NULL = 5;

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_open")]
      static extern int sqlite3_open(string filename, out IntPtr db);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_close")]
      static extern int sqlite3_close(IntPtr db);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_prepare_v2")]
      static extern int sqlite3_prepare_v2(IntPtr db, string zSql,
        int nByte, out IntPtr ppStmpt, IntPtr pzTail);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_step")]
      static extern int sqlite3_step(IntPtr stmHandle);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_finalize")]
      static extern int sqlite3_finalize(IntPtr stmHandle);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_errmsg")]
      static extern string sqlite3_errmsg(IntPtr db);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_column_count")]
      static extern int sqlite3_column_count(IntPtr stmHandle);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_column_origin_name")]
      static extern string sqlite3_column_origin_name(
        IntPtr stmHandle, int iCol);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_column_type")]
      static extern int sqlite3_column_type(IntPtr stmHandle, int iCol);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_column_int")]
      static extern int sqlite3_column_int(IntPtr stmHandle, int iCol);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_column_text")]
      static extern string sqlite3_column_text(IntPtr stmHandle, int iCol);

    [DllImport("sqlite3.dll", EntryPoint = "sqlite3_column_double")]
      static extern double sqlite3_column_double(IntPtr stmHandle, int iCol);
  }
}
```

basics, as with so many things in the open source community, it's already been done; navigate to switchonthecode.com/tutorials/csharp-tutorial-writing-a-dotnet-wrapper-for-sqlite, and we find a working set of P/Invoke declarations already laid down, shown in **Figure 1**.

The high fidelity of P/Invoke to C/C++ APIs makes this a relatively simple process—the SQLite API uses a raw pointer to represent the database itself, which P/Invoke sees as a System.IntPtr, and every so often the SQLite API uses a pointer to an int as a parameter so it can modify the contents, described by P/Invoke with the C# "out" keyword. (For more on P/Invoke, see pinvoke.codeplex.com.)

I'll refer you to the SQLite Web site for most of the details about how to use the SQLite API, but a quick glance at how to open a database, execute a query and then close the database would show you something like **Figure 2**.

This code, as can probably be inferred, opens the database around the file "persons.sqlite," prepares to create a table, executes that statement, cleans up that statement and then closes up the database. If for some reason the database fails to prepare or execute the query, an error message is harvested from the database (and can then be printed or logged or whatever else is needed).

I'm Feelin' Mighty Low

The most striking thing about this API is that it's a bit on the low-level side. If you're an old C++ hack like me, this may be a good thing, giving us a wonderful opportunity to reminisce about the Good Old Days, back when men were men, memory was managed by hand, and women swooned at cocktail parties over our scary stories of chasing down untamed pointers in the wilds of Windows 95 … but to the rest of these C# whipper-snappers, these Johnny-Come-Latelys who actually want to get productive work done, it's a bit too low to the ground, so to speak. What's needed is a good abstraction wrapper around that API to make it more manageable and cut down on the number of lines of code required to use it.

Wrapping this up into a single class isn't that difficult, particularly because System.Data provides some good classes that can handle most of the user-API interaction. Showing the full details of

Figure 2 **Opening a Database, Executing a Query and Closing the Database**

```
static void NativeMain()
 {
   // Open the database--db is our "handle" to it
   IntPtr db;
   if (SQLiteNative.sqlite3_open(@"cities.sqlite", out db)
     == SQLiteNative.SQLITE_OK)
     {
       // Prepare a simple DDL "CREATE TABLE" statement
       string query =
         "CREATE TABLE City " +
         "(name TEXT, state TEXT, population INTEGER)";
       IntPtr stmHandle;
       if (SQLiteNative.sqlite3_prepare_v2(db, query, query.Length,
         out stmHandle, IntPtr.Zero) != SQLiteNative.SQLITE_OK)
       {
         // Something went wrong--find out what
         var err = SQLiteNative.sqlite3_errmsg(db);
       }
       if (SQLiteNative.sqlite3_step(stmHandle) !=
         SQLiteNative.SQLITE_DONE)
       {
         // Something went wrong--find out what
         var err = SQLiteNative.sqlite3_errmsg(db);
       }
       if (SQLiteNative.sqlite3_finalize(stmHandle) !=
         SQLiteNative.SQLITE_OK)
       {
         // Something went wrong--find out what
         var err = SQLiteNative.sqlite3_errmsg(db);
       }

     // ... Now that we've created a table, we can insert some
     // data, query it back and so on

     // Close the database back up
     SQLiteNative.sqlite3_close(db);
     }
 }
```

that wrapper class, called SQLite, is a bit too lengthy to include here, but the declarations shown in **Figure 3** give a pretty clear indication of how it's supposed to be used.

Using it, then, would look something like the example in **Figure 4**.

Clearly there are more operations that could be added to the SQLite wrapper class in **Figure 4**, but it's already got the necessary barebones functionality, thanks in part to the wonderful database-agnostic nature of the core DataTable/DataRow/DataColumn classes in System.Data.

Abstractions, Abstractions

In some ways, the SQLite database's advantage is its low-level design and implementation—being embeddable means the "friction" involved in using it is pretty light. Add the wrapper classes, make sure the SQLite DLL is somewhere accessible to the program (typically by putting it into the directory with the executable) and now you're writing SQL statements like a champion. Assuming that's what you want to do, of course.

But it's likely that a significant majority of .NET Framework developers are either out of practice in managing a SQL database entirely "by hand" via console APIs, never knew how to do it or just want to leave that world behind. The modern .NET Framework environment provides such a wealth of tools for creating and managing relational schema that going back to this manual approach feels positively primitive and, more importantly, unproductive.

> It's relatively easy to create a wrapper class around the native declarations exposed in the SQLite DLL.

Furthermore, Microsoft has already spent effort creating an API that effectively describes most things a programmer wants to do against a relational database, and lots of those tools (LINQ to SQL, the Entity Framework and even the Visual Studio designer) are built on top of that API. I refer, of course, to ADO.NET and its provider model. Not having the ability to slide SQLite "underneath" ADO.NET means that all of that coolness is unavailable to the developer using SQLite, and that feels like a pretty significant shortcoming. The solution, then, is to build an ADO.NET provider for SQLite.

As we've already discovered, one of the nice things about the open source community is that there's a really good chance that whatever you're looking to do, somebody's already done it, and this is no different. System.Data.SQLite, available for download at sqlite.phxsoftware.com, is a full ADO.NET 3.5 provider, meaning that everything a developer can do with a traditional client/server relational database provider is available to the SQLite developer, including all the Visual Studio designer support as well as LINQ and the Entity Framework.

Using System.Data.SQLite is pretty straightforward. Grab the download (either the source, so you can build it yourself and poke around in the code to see how everything works—this is a good example to work from to learn how to build an ADO.NET provider, if you're curious—or grab just the binaries if you only want to get to "done" more quickly). Then drop the bits somewhere on your hard drive, reference System.Data.SQLite.dll from the project and life is good. Not surprisingly, the API classes live in System.Data.SQLite, and once they're referenced, you can write good ol' ADO.NET code against the database, as shown in **Figure 5**.

So far, so good. When the code is run from a Visual Studio 2005 or 2008 project, everything works flawlessly. But when the code is executed from Visual Studio 2010, an error comes up, claiming "Unhandled Exception: System.IO.FileLoadException: Mixed mode assembly is built against version 'v2.0.50727' of the runtime and cannot be loaded in the 4.0 runtime without additional configuration information." A mixed-mode assembly, for those

Figure 3 **Declarations of the SQLite Wrapper Class**

```
public class SQLite : IDisposable
 {
   private IntPtr _db; //pointer to SQLite database
   private bool _open; //whether or not the database is open

   /// <summary>
   /// Opens or creates SQLite database with the specified path
   /// </summary>
   /// <param name="path">Path to SQLite database</param>
   public void OpenDatabase(string path);

   /// <summary>
   /// Closes the SQLite database
   /// </summary>
   public void CloseDatabase();

   /// <summary>
   /// Executes a query that returns no results
   /// </summary>
   /// <param name="query">SQL query to execute</param>
   public void ExecuteNonQuery(string query);

   /// <summary>
   /// Executes a query and stores the results in
   /// a DataTable
   /// </summary>
   /// <param name="query">SQL query to execute</param>
   /// <returns>DataTable of results</returns>
   public DataTable ExecuteQuery(string query);
 }
```

Figure 4 **Using the SQLite Wrapper Class**

```
static void NativeWrapperMain()
{
  using (SQLite db = new SQLite("persons.sqlite"))
  {
    db.ExecuteNonQuery("CREATE Table Persons " +
      "(first TEXT, last TEXT, age INTEGER)");

    db.ExecuteNonQuery("INSERT INTO Persons (first, last, age) " +
      "VALUES ('Aaron', 'Erickson', 38)");
    db.ExecuteNonQuery("INSERT INTO Persons (first, last, age) " +
      "VALUES ('Rick', 'Minerich', 29)");
    db.ExecuteNonQuery("INSERT INTO Persons (first, last, age) " +
      "VALUES ('Talbott', 'Crowell', 35)");

    DataTable table = db.ExecuteQuery("SELECT * FROM Persons");

    foreach (DataRow row in table.Rows)
    {
      Console.WriteLine("{0} {1} {2}", row[0], row[1], row[2]);
    }
  }
}
```

who haven't heard the term before, is an assembly that contains both managed Microsoft Intermediate Language and native x86 assembly instructions. This, of course, is not good, on two levels—one, the obvious problem is we need to get the code to work, and two, if this is a mixed-mode assembly, it's going to create some problems when using SQLite in other environments, such as ASP.NET.

The first problem is easily solved by adding an app.config file that tells the CLR 4.0 to load the mixed-mode assembly:

```
<?xml version="1.0"encoding="utf-8" ?>
<configuration>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0"/>
  </startup>
</configuration>
```

A bigger problem is that a number of environments don't support mixed-mode assemblies, and in any event, there's a certain aesthetic involved here. For a variety of reasons, an all-managed solution would be preferable, but because the SQLite DLL is native code, that would be tricky. What would be nice is a port of the SQLite code base to C#, kept as close to the original C as possible.

All-Managed

Once again, the open source community provides when asked, and in this case, it provided a project called "C#-SQLite," available at code.google.com/p/csharp-sqlite. It apparently started as "an exercise to learn the C# language" by porting the code over, and the associated wiki has some discussion of what the author did to manage the port, but the upshot is that we now have exactly what we needed: an all-managed version of SQLite.

Using it requires downloading the project sources, opening the project and kicking off a build. Like a number of open source projects, C#-SQLite consists of several projects, but each one is enclosed in its own solution file, so you may need to open more than one solution. (Or just kick off the builds from the command line with MSBuild—whatever works best.)

Once it's built, add the C#-SQLite assembly (Community.CSharpSQLite) to the project, and for ADO.NET support, add the C#-SQLite Client assembly (Community.CsharpSqlite.SQLiteClient.dll) as well. Once again, the full capabilities of SQLite are available to us through an ADO.NET provider, such that we can rewrite almost the exact same code shown earlier (see **Figure 6**).

Figure 5 **Using System.Data.SQLite**

```
static void ManagedWrapperMain()
{
  var connStr = new SQLiteConnectionStringBuilder()
    { DataSource = "persons.sqlite" };
  using (SQLiteConnection conn = new SQLiteConnection(connStr.ToString()))
  {
    conn.Open();
    SQLiteCommand cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT COUNT(*) FROM Persons";
    var ct = cmd.ExecuteScalar();
    Console.WriteLine("Count = {0}", ct);

    cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT * FROM Persons";
    SQLiteDataReader reader = cmd.ExecuteReader();
    DataTable dt = new DataTable();
    dt.Load(reader);
    foreach (DataRow row in dt.Rows)
    {
      Console.WriteLine("{0} {1} {2}", row[0], row[1], row[2]);
    }
  }
}
```

Figure 6 **Using C#-SQLite**

```
Static void AllManagedMain()
{
  SqliteConnection conn =
    New SqliteConnection(@"Version=3,uri=file:persons.sqlite");
  conn.Open();
  try
  {
    SqliteCommand cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT COUNT(*) FROM Persons";
    var ct = cmd.ExecuteScalar();
    Console.WriteLine("Count = {0}", ct);

    cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT * FROM Persons";
    SqliteDataReader reader = cmd.ExecuteReader();
    while (reader.Read())
    {
      Console.WriteLine("{0} {1} {2}", reader[0], reader[1], reader[2]);
    }
  }
  finally
  {
    conn.Close();
  }
}
```

Notice how the APIs are almost identical to the earlier mixed-mode version (only the class names have changed, and even then it's really just a question of case: "SQLite" vs. "Sqlite" as a prefix, for example), but now we get all of the SQLite goodness without the potential security concerns (if any) of a native-mode DLL.

Limitations

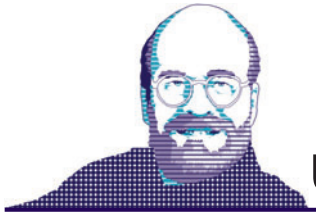
Despite the wonderful nature of SQLite, it's important to understand its limitations if the decision between using SQLite and SQL Server is to be made with any degree of sanity. SQLite is not going to provide all the features of SQL Server—far from it. The SQLite database doesn't even want developers using it to use multiple threads, much less access it from multiple threads. In fact, it's fair to say that if two programs want access to a SQLite database simultaneously, it's probably time to upgrade to a SQL Server instance (Express or otherwise).

SQLite's principal areas of "win" will be in many of the same areas that Access files used to occupy, with a near-complete SQL-92 syntax to back it, along with an ability to read database files used by other environments (Python, Perl and so on). Using it from Silverlight or phone clients is also a highly interesting area, particularly for local storage—sticking a SQLite database into Silverlight isolated storage would give developers a portable (in that it can travel with the Silverlight code) database in which to store local data, for example. Use it judiciously, and SQLite rounds out a relational database continuum of functionality-to-weight options.

Again, if there's a particular topic you'd like to see explored, don't hesitate to drop me a note. In a very real way, it's your column, after all.

Happy coding! ■

TED NEWARD *is a principal with Neward & Associates, an independent firm specializing in enterprise Microsoft .NET Framework and Java platform systems. He has written more than 100 articles, is a C# MVP, INETA speaker and the author and coauthor of a dozen books, including "Professional F# 2.0" (Wrox 2010). He consults and mentors regularly. Reach him at ted@tedneward.com and read his blog at blogs.tedneward.com.*



CHARLES PETZOLD

Multi-Touch Manipulation Events in WPF

Just within the past few years, multi-touch has progressed from a futuristic sci-fi film prop to a mainstream UI. Multi-touch displays are now standard on new models of smartphones and tablet computers. Multi-touch is also likely to become ubiquitous on computers in public spaces, such as kiosks or the table computer pioneered by Microsoft Surface.

The only real uncertainly is the popularity of multi-touch on the conventional desktop computer. Perhaps the greatest impediment is the fatigue known as "gorilla arm" associated with moving fingers on vertical screens for long periods of time. My personal hope is that the power of multi-touch will actually provoke a redesign of the desktop display. I can envision a desktop computer with a display resembling the configuration of a drafting table, and perhaps almost as large.

But that's the future (perhaps). For the present, developers have new APIs to master. The support for multi-touch in Windows 7 has filtered down and settled into various areas of the Microsoft .NET Framework with interfaces both low and high.

Sorting out the Multi-Touch Support

If you consider the complexity of expression that's possible with the use of multiple fingers on a display, you can perhaps appreciate why nobody seems to know quite yet the "correct" programming interface for multi-touch. This will take some time. Meanwhile, you have several options.

Windows Presentation Foundation (WPF) 4.0 has two multi-touch interfaces available for programs running under Windows 7. For specialized uses of multi-touch, programmers will want to explore the low-level interface consisting of several routed events defined by UIElement named TouchDown, TouchMove, TouchUp, TouchEnter, TouchLeave, with preview versions of the down, move and up events. Obviously these are modeled after the mouse events, except that an integer ID property is necessary to keep track of multiple fingers on the display. Microsoft Surface is built on WPF 3.5, but it supports a more extensive low-level Contact interface that distinguishes types and shapes of touch input.

The subject of this column is the high-level multi-touch support in WPF 4.0, which consists of a collection of events whose names begin with the word Manipulation. These Manipulation events perform several crucial multi-touch jobs:

- consolidating the interaction of two fingers into a single action
- resolving the movement of one or two fingers into transforms
- implementing inertia when the fingers leave the screen

Figure 1 **The Manipulation Events in Windows Presentation Foundation 4.0**

Event	Supported by Windows Phone 7?
ManipulationStarting	No
ManipulationStarted	Yes
ManipulationDelta	Yes
ManipulationInertiaStarted	No
ManipulationBoundaryFeedback	No
ManipulationCompleted	Yes

A subset of the Manipulation events is listed in the documentation of Silverlight 4, but that's a bit deceptive. The events are not yet supported by Silverlight itself, but they are supported in Silverlight applications written for Windows Phone 7. The Manipulation events are listed in **Figure 1**.

Web-based Silverlight 4 applications will continue to use the Touch.FrameReported event that I discussed in the article "Finger Style: Exploring Multi-Touch Support in Silverlight" in the March 2010 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee336026).

Along with the Manipulation events themselves, the UIElement class in WPF also supports overridable methods such as OnManipulationStarting corresponding to the Manipulation events. In Silverlight for Windows Phone 7, these overridable methods are defined by the Control class.

A Multi-Touch Example

Perhaps the archetypal multi-touch application is a photograph viewer that lets you move photos on a surface, make them larger or smaller with a pair of fingers, and rotate them. These operations are sometimes referred to as pan, zoom and rotate, and they correspond to the standard graphics transforms of translation, scaling and rotation.

Obviously a photograph-viewing program needs to maintain the collection of photos, allow new photos to be added and photos to be removed, and it's always nice to display the photos in a little graphical frame, but I'm going to ignore all that and just focus on the multi-touch interaction. I was surprised how easy it all becomes with the Manipulation events, and I think you will be as well.

All the source code for this column is in a single downloadable solution named WpfManipulationSamples. The first project is

Code download available at code.msdn.microsoft.com/mag201008UF.

Figure 2 **The XAML File for SimpleManipulationDemo**

```
<Window x:Class="SimpleManipulationDemo.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Simple Manipulation Demo">

  <Window.Resources>
    <Style TargetType="Image">
      <Setter Property="Stretch" Value="None" />
      <Setter Property="HorizontalAlignment" Value="Left" />
      <Setter Property="VerticalAlignment" Value="Top" />
    </Style>
  </Window.Resources>

  <Grid>
    <Image Source="Images/112-1283_IMG.JPG"
      IsManipulationEnabled="True"
      RenderTransform="0.5 0 0 0.5 100 100" />

    <Image Source="Images/139-3926_IMG.JPG"
      IsManipulationEnabled="True"
      RenderTransform="0.5 0 0 0.5 200 200" />

    <Image Source="Images/IMG_0972.JPG"
      IsManipulationEnabled="True"
      RenderTransform="0.5 0 0 0.5 300 300" />

    <Image Source="Images/IMG_4675.JPG"
      IsManipulationEnabled="True"
      RenderTransform="0.5 0 0 0.5 400 400" />
  </Grid>
</Window>
```

SimpleManipulationDemo, and the MainWindow.xaml file is shown in **Figure 2**. The Grid contains four Image elements.

First notice the setting on all three Image elements:

```
IsManipulationEnabled="True"
```

This property is false by default. You must set it to true for any element on which you want to obtain multi-touch input and generate Manipulation events.

The Manipulation events are WPF routed events, meaning that the events bubble up the visual tree. In this program, neither the Grid nor MainWindow have the IsManipulationEnabled property set to true, but you can still attach handlers for the Manipulation events to the Grid and MainWindow elements, or override the OnManipulation methods in the MainWindow class.

Notice also that each of the Image elements has its RenderTransform set to a six-number string:

```
RenderTransform="0.5 0 0 0.5 100 100"
```

This is a shortcut that sets the RenderTransform property to an initialized MatrixTransform object. In this particular case, the Matrix object set to the MatrixTransform is initialized to perform a scale of 0.5 (making the photos half their actual size) and a translation of 100 units to the right and down. The code-behind file for the window accesses and modifies this MatrixTransform.

The complete MainWindow.xaml.cs file is shown in **Figure 3**, and overrides just two methods, OnManipulationStarting and OnManipulationDelta. These methods process the manipulations generated by the Image elements.

Manipulation Basics

A manipulation is defined as one or more fingers touching a particular element. A complete manipulation begins with the ManipulationStarting event (followed soon thereafter by ManipulationStarted) and ends with ManipulationCompleted. In between, there might be many ManipulationDelta events.

Each of the Manipulation events is accompanied by its own set of event arguments encapsulated in a class named after the event with EventArgs appended, such as ManipulationStartingEventArgs and ManipulationDeltaEventArgs. These classes derive from the familiar InputEventArgs, which in turn derives from RoutedEventArgs. The classes include Source and OriginalSource properties indicating where the event originated.

In the SimpleManipulationDemo program, Source and OriginalSource will both be set to the Image element generating the Manipulation events. Only an element with its IsManipulation-

Figure 3 **The Code-Behind File for SimpleManipulationDemo**

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace SimpleManipulationDemo {
  public partial class MainWindow : Window {
    public MainWindow() {
      InitializeComponent();
    }

    protected override void OnManipulationStarting(
      ManipulationStartingEventArgs args) {

      args.ManipulationContainer = this;

      // Adjust Z-order
      FrameworkElement element =
        args.Source as FrameworkElement;
      Panel pnl = element.Parent as Panel;

      for (int i = 0; i < pnl.Children.Count; i++)
        Panel.SetZIndex(pnl.Children[i],
          pnl.Children[i] ==
          element ? pnl.Children.Count : i);

      args.Handled = true;
      base.OnManipulationStarting(args);
    }

    protected override void OnManipulationDelta(
      ManipulationDeltaEventArgs args) {

      UIElement element = args.Source as UIElement;
      MatrixTransform xform =
        element.RenderTransform as MatrixTransform;
      Matrix matrix = xform.Matrix;
      ManipulationDelta delta = args.DeltaManipulation;
      Point center = args.ManipulationOrigin;

      matrix.ScaleAt(
        delta.Scale.X, delta.Scale.Y, center.X, center.Y);
      matrix.RotateAt(
        delta.Rotation, center.X, center.Y);
      matrix.Translate(
        delta.Translation.X, delta.Translation.Y);
      xform.Matrix = matrix;

      args.Handled = true;
      base.OnManipulationDelta(args);
    }
  }
}
```

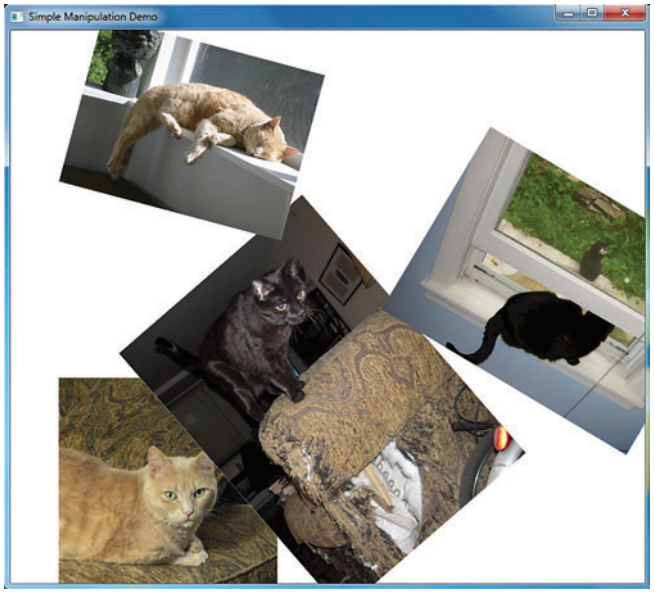


Figure 4 **The SimpleManipulationDemo Program**

Enabled property set to true will show up as the Source and OriginalSource properties in these Manipulation events.

In addition, each of the event argument classes associated with the Manipulation events includes a property named ManipulationContainer. This is the element within which the multi-touch manipulation is occurring. All coordinates in the Manipulation events are relative to this container.

By default, the ManipulationContainer property is set to the same element as the Source and OriginalSource properties—the element being manipulated—but this is probably *not* what you want. In general, you don't want the manipulation container to be the same as the element being manipulated because tricky interactions get involved with dynamically moving, scaling and rotating the same element that's reporting touch information. Instead, you want the manipulation container to be a parent of the manipulated element, or perhaps an element further up the visual tree.

In most of the Manipulation events, the ManipulationContainer property is get-only. The exception is the very first Manipulation event that an element receives. In ManipulationStarting you have the opportunity to change ManipulationContainer to something more appropriate. In the SimpleManipulationDemo project, this job is a single line of code:

```
args.ManipulationContainer = this;
```

In all subsequent events, ManipulationContainer will then be the MainWindow element rather than the Image element, and all coordinates will be relative to that window. This works fine because the Grid containing the Image elements is also aligned with the window.

The remainder of the OnManipulationStarting method is dedicated to bringing the touched Image element to the foreground by resetting the Panel.ZIndex attached properties of all the Image elements in the Grid. This is a simple way of handling ZIndex but probably not the best because it creates sudden changes.

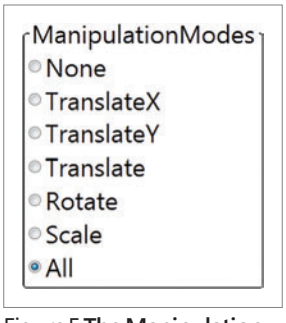


Figure 5 **The ManipulationModeDemo Display**

ManipulationDelta and DeltaManipulation

The only other event handled by SimpleManpulationDemo is ManipulationDelta. The ManipulationDeltaEventArgs class defines two properties of type ManipulationDelta. (Yes, the event and the class have the same name.) These properties are DeltaManipulation and CumulativeManipulation. As the names suggest, DeltaManipulation reflects the manipulation that occurred since the last ManipulationDelta event, and CumulativeManipulation is the complete manipulation that began with the ManipulationStarting event.

ManipulationDelta has four properties:

- Translation of type Vector
- Scale of type Vector
- Expansion of type Vector
- Rotation of type double

The Vector structure defines two properties named X and Y of type double. One of the more significant differences with the Manipulation support under Silverlight for Windows Phone 7 is the absence of the Expansion and Rotation properties.

The Translation property indicates movement (or a pan) in the horizontal and vertical directions. A single finger on an element can generate changes in translation, but translation can also be part of other manipulations.

The Scale and Expansion properties both indicate a change in size (a zoom), which always requires two fingers. Scale is multiplicative and Expansion is additive. Use Scale for setting a scale transform; use Expansion for increasing or decreasing the Width and Height properties of an element by device-independent units.

In WPF 4.0, the X and Y values of the Scale vector are always the same. The Manipulation events do not give you sufficient information to scale an element anisotropically (that is, differently in the horizontal and vertical directions).

By default, Rotation also requires two fingers, although you'll see later how to enable one-finger rotation. In any particular ManipulationDelta event, all four properties might be set. A pair of fingers might be enlarging an element, and at the same time rotating it and moving it to another location.

Scaling and rotation are always relative to a particular center point. This center is also provided in ManipulationDeltaEventArgs in the property named ManipulationOrigin of type Point. This origin is relative to the ManipulationContainer set in the ManipulationStarting event.

Your job in the ManipulationDelta event is to modify the RenderTransform property of the manipulated object in accordance with the delta values in the following order: scaling first, then rotation, and finally translation. (Actually, because the horizontal and vertical scaling factors are identical, you can switch the order of the scaling and rotation transforms and still get the same result.)

The OnManipulationDelta method in **Figure 3** shows a standard approach. The Matrix object is obtained from the MatrixTransform set on the manipulated Image element. It's modified through calls to ScaleAt and RotateAt (both relative to the ManipulationOrigin) and Translate. Matrix is a

structure rather than a class, so you must finish up by replacing the old value in the MatrixTransform with the new one.

It's possible to vary this code a little. As shown, it scales around a center with this statement:

```
matrix.ScaleAt(delta.Scale.X, delta.Scale.Y, center.X, center.Y);
```

This is equivalent to translating to the negative of the center point, scaling and then translating back:

```
matrix.Translate(-center.X, -center.Y);
matrix.Scale(delta.Scale.X, delta.Scale.Y);
matrix.Translate(center.X, center.Y);
```

The RotateAt method can likewise be replaced with this:

```
matrix.Translate(-center.X, -center.Y);
matrix.Rotate(delta.Rotation);
matrix.Translate(center.X, center.Y);
```

The two adjacent Translate calls now cancel each other out, so the composite is:

```
matrix.Translate(-center.X, -center.Y);
matrix.Scale(delta.Scale.X, delta.Scale.Y);
matrix.Rotate(delta.Rotation);
matrix.Translate(center.X, center.Y);
```

It's probably a little bit more efficient.

Figure 4 shows the SimpleManipulationDemo program in action.

Enabling the Container?

One of the interesting features of the SimpleManpulationDemo program is that you can simultaneously manipulate two Image elements, or even more if you have the hardware support and a sufficient number of fingers. Each Image element generates its own ManipulationStarting event and its own series of ManipulationDelta events. The code effectively distinguishes between the multiple Image elements by the Source property of the event arguments.

For this reason, it's important not to set any state information in fields that implies that only one element can be manipulated at a time.

The simultaneous manipulation of multiple elements is possible because each of the Image elements has its own IsManipulationEnabled property set to true. Each of them can generate a unique series of Manipulation events.

When approaching these Manipulation events for the first time, you might instead investigate setting IsManpulationEnabled to true only on the MainWindow class or another element serving as a container. This is possible, but it's somewhat clumsier in practice and not quite as powerful. The only real advantage is that you don't need to set the ManipulationContainer property in the ManipulationStarting event. The messiness comes later when you must determine which element is being manipulated by hit-testing on the child elements using the ManipulationOrigin property in the ManipulatedStarted event.

You would then need to store the element being manipulated as a field for use in future ManipulationDelta events. In this case, it's safe to store state information in fields because you'll only be able to manipulate one element in the container at a time.

The Manipulation Mode

As you saw, one of the crucial properties to set during the ManipulationStarting event is the ManipulationContainer. Another couple of properties are useful to customize the particular manipulation.

You can limit the types of manipulation you can perform by initializing the Mode property with a member of the ManipulationModes enumeration. For example, if you were using manipulation solely for scrolling horizontally, you might want to limit the events to just horizontal translation. The ManipulationModesDemo program lets you set the mode dynamically by displaying a list of RadioButton elements listing the options, as shown in **Figure 5**.

Of course, the RadioButton is one of the many controls in WPF 4.0 that respond directly to touch.

The Single Finger Rotation

By default, you need two fingers to rotate an object. However, if a real photo is sitting on a real desk, you can put your finger on the corner and rotate it in a circle. The rotation is roughly occurring around the center of the object.

You can do this with the Manipulation events by setting the Pivot property of ManipulationStartingEventArgs. By default the Pivot property is null; you enable one-finger rotation by setting the property to a ManipulationPivot object. The key property of ManipulationPivot is Center, which you might consider calculating as the center of the element being manipulated:

```
Point center = new Point(element.ActualWidth / 2,
                         element.ActualHeight / 2);
```

But this center point must be relative to the manipulation container, which in the programs I've been showing you is the element handling the events. Translating that center point from the element being manipulated to the container is easy:

```
center = element.TranslatePoint(center, this);
```

Another little piece of information also needs to be set. If all you're specifying is a center point, a problem arises when you put your finger right in the center of the element: just a little movement will cause the element to spin around like crazy! For this reason, ManipulationPivot also has a Radius property. Rotation will not occur if the finger is within Radius units of the Center point. The ManipulationPivotDemo program sets this radius to half an inch:

```
args.Pivot = new ManipulationPivot(center, 48);
```

Now a single finger can perform a combination of rotation and translation.

Beyond the Basics

What you've seen here are the basics of using the WPF 4.0 Manipulation events. Of course, there are some variations on these techniques that I'll show in future columns, as well as the power of manipulation inertia.

You may also want to take a look at the Surface Toolkit for Windows Touch (msdn.microsoft.com/library/ee957352), which provides touch-optimized controls for your apps. The ScatterView control in particular eliminates the need for using the Manipulation events directly for basic stuff like manipulating photos. It has some snazzy effects and behaviors that will make sure your app behaves the same as other touch apps. ■

CHARLES PETZOLD *is a longtime contributing editor to* MSDN Magazine. *He's currently writing "Programming Windows Phone 7," which will be published as a free downloadable e-book in the fall of 2010. A preview edition is currently available through his Web site, charlespetzold.com.*

THANKS to the following technical experts for reviewing this article: *Doug Kramer, Robert Levy and Anson Tsao*

Mars and Venus

A client of mine was scratching his head over his wife's behavior when he'd taken her shopping for a new netbook. "Look at this, honey," he had told her. "A dual-core processor. How'd they get one into this little box? Wow, what a neato heat sink."

"That's nice, dear," she said to him. "But will it do Facebook? That's what I want."

"I told her all about the disk and the memory, how it was a really nice unit for the price," the guy told me, "But all she kept asking was, 'Will it do Facebook?' How can she not care about this important stuff?"

I don't think he'd been married very long.

My client was reporting a common, almost archetypal, division of thought, which market research guru Paco Underhill discussed in his book "Why We Buy: The Science of Shopping" (Simon & Schuster, 1999). "Men are in love with the technology itself, with the gee-whiz factor, with the horsepower. … [They're] gathered around the barbecue comparing the size of their hard drives and the speed of their modems. As they say, it's a dude thing." Women, on the other hand, "take a completely different approach to the world of high-tech. They take technologies and turn them into appliances. They strip even the fanciest gizmo of all that is mysterious and jargony in order to determine its usefulness. Women look at technology and see its purpose, its reason—what it can do [for them]. The promise of technology is always that it will make our lives easier and more efficient. Women are the ones who demand that it fulfill its purpose." The husband in my example obsessed over hardware—the means. The wife concentrated on Facebook—the desired end.

> Half of your users are female, which means that their thought processes probably resemble those of my client's wife more than they do yours.

The developer population is almost entirely male. It reached 25 percent female at one point; now it's down to 10 percent in the United States, and less in Europe. The user population was also once all-male, back when the Earth was still cooling off and the last few dinosaurs tottered around wondering where their friends had gone; today it isn't. Half of your users are female, which means that their thought processes probably resemble those of my client's wife more than they do yours.

And it's often more than half. Many user populations contain far more women than men—nurses, for example (94 percent female, according to nursingadvocacy.org/faq/rn_facts.html) or elementary school teachers (85 percent female, according to a bnet.com article: tinyurl.com/26xub2p). I recently walked through the customer service department of one of my financial-industry clients and counted 100 percent females, mostly in their 40s and 50s, soon to be using software built by men in their 20s. At least I got those guys thinking about the differences.

What pleases a female user? If I knew for sure, I'd be retired on my private island, not knocking out these columns for a pittance per word. I remember trying to convince my mother how great my idea was of self-location and directions on her cell phone (back in 2004, before this became ubiquitous). She wasn't interested. "I can usually figure out where I am without much trouble, and if not, I ask someone. I know you want me to say yes because you think it's so cool, but I don't really care. Sorry." I had a flash of sympathy for the geekiest guy in the galaxy when Harry Mudd told him: "You're an excellent science officer, Mr. Spock, but you couldn't sell false patents to your mother."

When we're trying to develop software that doesn't suck, we have to consider the difference between an X and a Y chromosome. It's bigger than you think. Next time you're roughing out a design, ask your mother what she would do with it. And listen to what she tells you.

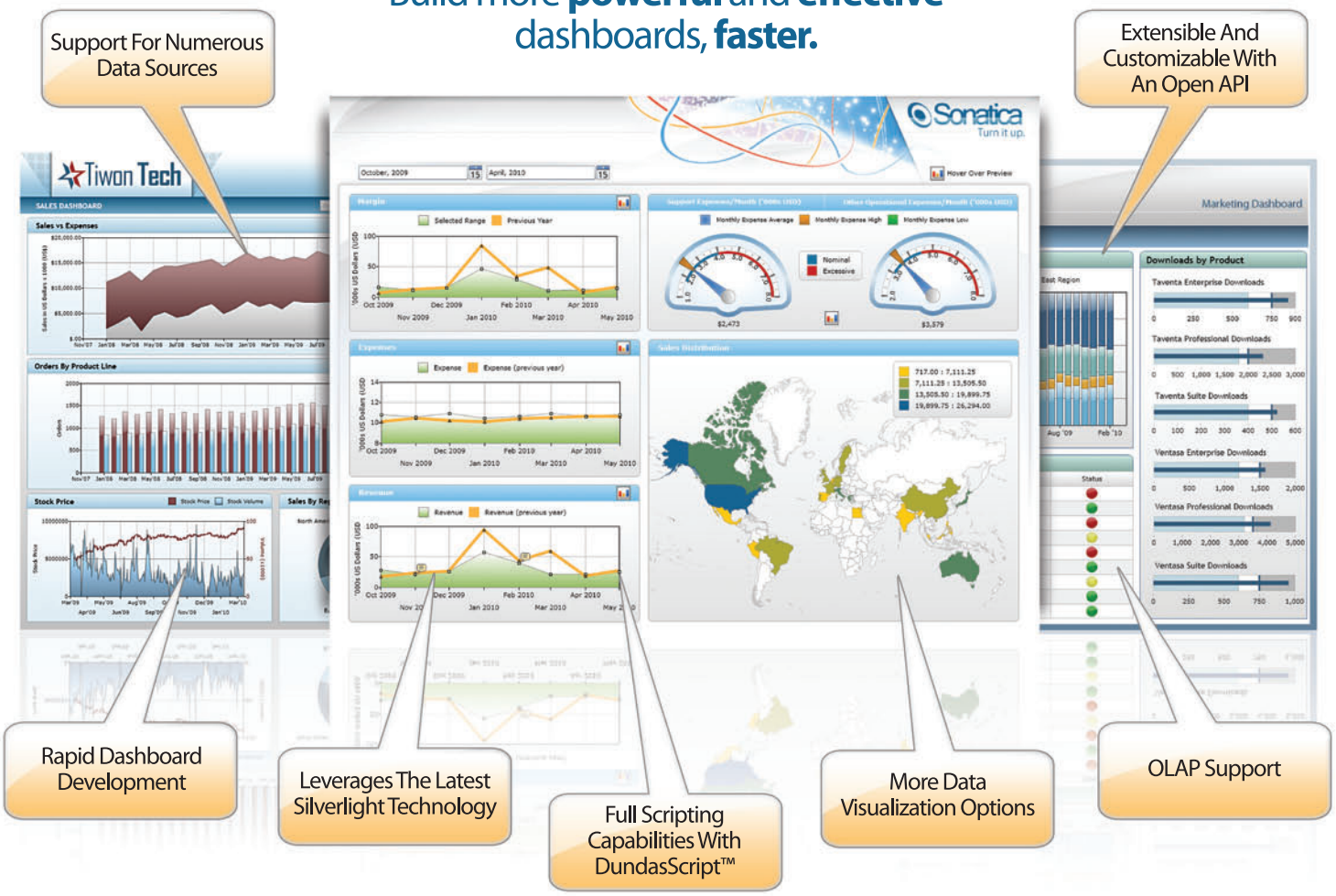
Reader mail department: In response to my statement in the May issue that "your mother might [be interested in your software], because you wrote it and she loves you," a reader named Barclay drew my attention to B. B. King's classic blues song, "Nobody Loves Me But My Mother, And She Could Be Jivin' Too." You'll find it online in the usual places, such as youtube.com/watch?v=OIW4ARVbhrw.

DAVID S. PLATT *teaches Programming .NET at Harvard University Extension School and at companies all over the world. He is the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

Dundas Dashboard

The Dashboard Platform for Developers like you

Build more **powerful** and **effective** dashboards, **faster.**



Dundas Dashboard is a ground-breaking, extensible solution that uses a revolutionary approach to dashboard creation, providing you with a unified view of key metrics and a new level of strategic insight and decision-making.

Powered by Microsoft® Silverlight™

Dundas Data Visualization

www.dundas.com/dashboard

(416) 467-5100 • (800) 463-1492