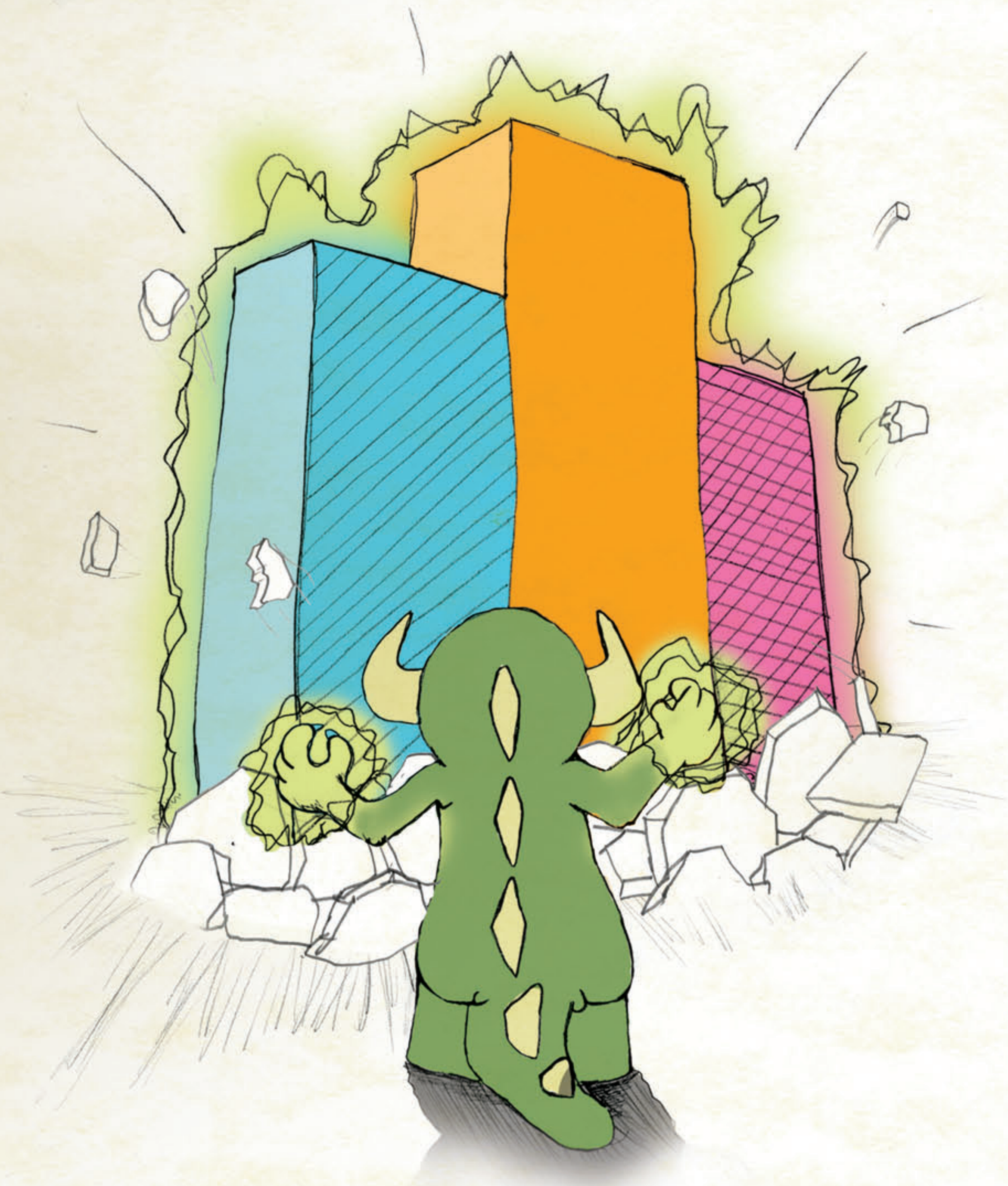
THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2010 VOL 25 NO 5

msdn magazine

Microsoft



BREAK OUT OF THE BOX

Sure, Visual Studio 2010 has a lot of great functionality—we're excited that it's only making our User Interface components even better! We're here to help you go beyond what Visual Studio 2010 gives you so you can create Killer Apps quickly, easily and without breaking a sweat! Go to infragistics.com/beyondthebox today to expand your toolbox with the fastest, best-performing and most powerful UI controls available. You'll be surprised by your own strength!

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

Infragistics®

KILLER APPS. NO EXCUSES.

dtSearch®

Instantly Search Terabytes of Text

- 25+ full-text and fielded data search options
- Built-in file parsers and converters highlight hits in popular file types
- Spider supports static and dynamic web data; highlights hits with links, formatting and images intact
- API supports C++, .NET, Java, SQL, etc. .NET Spider API. Includes 64-bit (Win/Linux)
- Fully-functional evaluations available

Content extraction only licenses also available

"Bottom line: dtSearch manages a terabyte of text in a single index and returns results in less than a second" — InfoWorld

dtSearch "covers all data sources … powerful Web-based engines" — eWEEK

"Lightning fast ... performance was unmatched by any other product" — Redmond Magazine

For hundreds more reviews, and hundreds of developer case studies, see www.dtSearch.com

1-800-IT-FINDS • www.dtSearch.com

The Smart Choice for Text Retrieval® since 1991

msdn® magazine

MAY 2010 VOLUME 25 NUMBER 5

LUCINDA ROWLEY Director
DIEGO DAGUM Editorial Director
KERI GRASSL Site Manager

KEITH WARD Editor in Chief
TERRENCE DORSEY Technical Editor
DAVID RAMEL Features Editor
WENDY GONCHAR Managing Editor
MARTI LONGWORTH Associate Managing Editor

SCOTT SHULTZ Creative Director
JOSHUA GOULD Art Director
ALAN TAO Senior Graphic Designer

CONTRIBUTING EDITORS K. Scott Allen, Dino Esposito, Julie Lerman, Juval Lowy, Dr. James McCaffrey, Ted Neward, Charles Petzold, David S. Platt

RedmondMediaGroup

Henry Allain President, Redmond Media Group
Matt Morollo Vice President, Publishing
Doug Barney Vice President, Editorial Director
Michele Imgrund Director, Marketing
Tracy Cook Online Marketing Director

ADVERTISING SALES: 508-532-1418/mmorollo@1105media.com

Matt Morollo VP, Publishing
Chris Kourtoglou Regional Sales Manager
William Smith National Accounts Director
Danna Vedder Microsoft Account Manager
Jenny Hernandez-Asandas Director Print Production
Serena Barnes Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA INC

Neal Vitale President & Chief Executive Officer
Richard Vitale Senior Vice President & Chief Financial Officer
Michael J. Valenti Executive Vice President

Abraham M. Langer Senior Vice President, Audience Development & Digital Media
Christopher M. Coates Vice President, Finance & Administration
Erik A. Lindgren Vice President, Information Technology & Application Development
Carmel McDonagh Vice President, Attendee Marketing
David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

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, P.O. Box 3167, Carol Stream, IL 60132, e-mail MSDNmag@1105service.com or call 847-763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, P.O. 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

Microsoft® Visual Studio®

Microsoft®

/*IT'S EVERYWHERE.*/

Code. It's used to create the things that are all around us. Almost everywhere you look, it's there. Just like the possibilities you see as a developer. With Visual Studio 2010, you can realize your vision with new tools that will dramatically impact the way you work, from design to development to deployment.

LIFE RUNS ON CODE.
WHAT WILL YOU DO WITH VISUAL STUDIO 2010?

Get started at **VisualStudio.com**

SNAP IT

[Snap this tag to get the latest news on Visual Studio 2010 or Text VS2010 to 21710*]
Get the free app for your phone at http://gettag.mobi
*Standard messaging and data charges apply.

This Way-Cool 'Internet' Doohickey

Hey, have you heard about this groovy new thing called the "Internet"? Some people call it the "World Wide Web," or "Web" for short, although I think the Web is just part of it.

Anyway, it's a way that your computer connects to other computers. It's mostly done by visiting these things called "Web sites." Web sites can be almost anything—places to chat with friends, stores that only exist on computers (special ones called "servers"), newspapers that you can read with a computer or even just some guy who has "posted" tons of stuff about that TV show "Baywatch." When you're there, it's called being "online." You can buy books and clothes and cars and even trade junk with other people.

I've even heard that eventually stuff like video and audio will be available on this Internet thingy. Pretty cool, huh?

Rich Internet Applications, like Silverlight 4, continue to push forward like Lewis and Clark, exploring new ways to build and deliver apps.

Bet you haven't had a conversation like that in a while, eh? In reality (you know, non-geek years), the Internet—as a commercial and social medium—hasn't been around all that long. I remember working as a news director at a Baltimore TV station's Web site in 2000. It was my first real exposure to the day-to-day workings of the Web. I remember learning about things called "HTML" and "Cascading Style Sheets" and "alt tags" and wondering what the heck I was getting myself into.

Now here we are a decade later, and it's hard to remember the days when I got my news from the plastic-wrapped paper deposited on my front yard every morning. Or had to drive to a bookstore to get the latest Stephen King novel. Portable phones were brick-sized (and -weight) devices with long antennas, and you couldn't do anything with them but call people.

Those days now belong to the archives, and that Internet thingy continues to mature and grow. Rich Internet Applications, like Silverlight 4, continue to push forward like Lewis and Clark, exploring new ways to build and deliver apps. That's our focus this month.

Product Reviews from Readers, for Readers

In much the same way, *MSDN Magazine* is similarly moving ahead. We're going to be adding something special to both the print and online versions of *MSDN Magazine*, starting in a few months. And we want you—need you, in fact—to be a part of it.

We're going to start running product reviews written by our readers. So if you're a developer using something you think is exceptionally cool, let us know about how you use it and what it does for you. On the other hand, if you're using something that just doesn't work, or work well, that's valuable information, too. We'd love to hear from you.

We're looking for articles in the range of 500 to 1,000 words. The products can be from Microsoft or another source—anything you use to help you in your job. We're looking for honest, open and clear reviews.

You don't have to be a published author to write a review. We're looking for passionate developers who have something interesting to say and want to share their experiences with their colleagues.

Authors will be paid for articles. The reviews will appear both in the print issue of the magazine and the online version.

One thing to note: we *will* verify the authenticity of all authors and products. That means if you're a vendor posing as an author, with the goal of touting your latest ground-breaking product by disguising it as a review, we'll find out about it. Remember that these reviews are only as valuable as the honesty of the reviewer, so please avoid any of those types of shenanigans.

For more information about the process, and guidelines for the review, contact us at mmeditor@microsoft.com. Make sure you put "Reader Product Reviews" in the subject line, so we'll be sure to flag it.

We're looking forward to hearing from you!

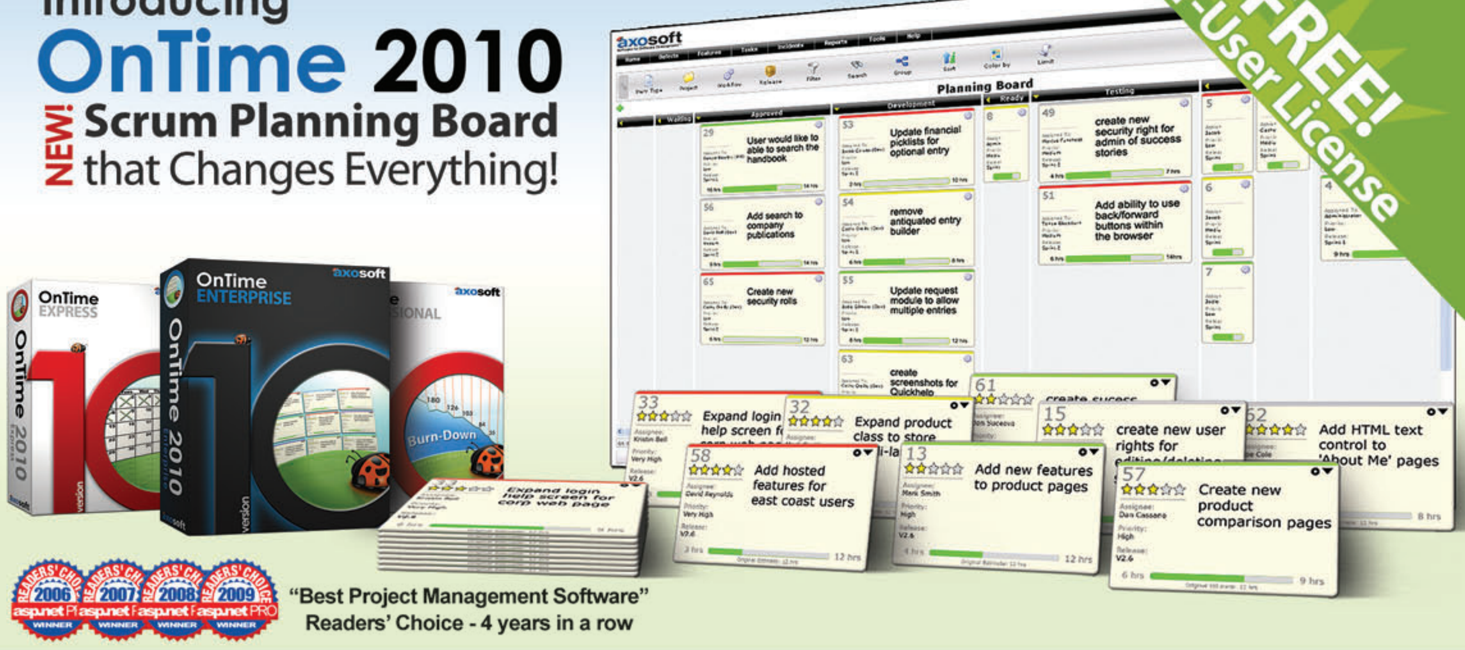
Keith Ward

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!



"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.

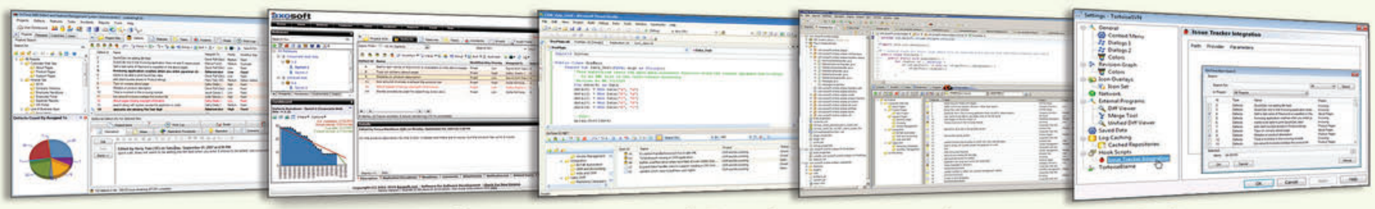
Workflow
Define a hyper-fast, lightweight, agile workflow or a robust process-enforcement system.

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.



axosoft

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

800.653.0024
www.axosoft.com

Using the Dynamic Keyword in C# 4.0

The introduction of static type checking represented an important milestone in the history of programming languages. In the 1970s, languages such as Pascal and C started enforcing static types and strong type checking. With static type checking, the compiler will produce an error for any call that fails to pass a method argument of the appropriate type. Likewise, you should expect a compiler error if you attempt to call a missing method on a type instance.

Other languages that push forward the opposite approach—dynamic type checking—have come along over the years. Dynamic type checking contradicts the idea that the type of a variable has to be statically determined at compile time and can never change while the variable is in scope. Note, however, that dynamic type checking doesn't confer wholesale freedom to mix types, pretending they're the same. For example, even with dynamic type checking, you still can't add a Boolean value to an integer. The difference with dynamic type checking is that the check occurs when the program executes rather than when it compiles.

Statically Typed or Dynamically Typed

Visual Studio 2010 and C# 4.0 provide a new keyword, dynamic, that enables dynamic typing in what has traditionally been a statically typed language. Before diving into the dynamic aspects of C# 4.0, though, we need to get some basic terminology down.

Let's define a variable as a storage location that's restricted to values of a particular type. Next, let's specify four fundamental properties of a statically typed language:

- Every expression is of a type known at compile time.
- Variables are restricted to a type known at compile time.
- The compiler guarantees that type restrictions on assignments of expressions into variables meet the restrictions on the variables.
- Semantic analysis tasks, such as overload resolution, occur at compile time and the results are baked into the assembly.

A dynamic language has the opposite properties. Not every expression is of a known type at compile time, nor is every variable. Storage restrictions, if any, are checked at run time and ignored at compile time. Semantic analysis occurs only at run time.

A statically typed language does let you make some operations dynamic. The cast operator exists so you can attempt a type conversion as a runtime operation. The conversion is part of the program code, and you can summarize the semantic expressed by the cast operator as "dynamically check the validity of this conversion at run time."

However, concerning attributes such as dynamic and static (or perhaps strong and weak): Today they're better applied to individual features of a programming language than to the language as a whole.

Let's briefly consider Python and PHP. Both are dynamic languages, let you use variables, and allow the runtime environment to figure out the actual type stored in it. But with PHP you can store, say, integers and strings in the same variable in the same scope. In this regard, PHP (like JavaScript) is a weakly typed, dynamic language.

On the other hand, Python gives you only one chance to set the type of a variable, which makes it more strongly typed. You can dynamically assign the type to a variable and have the runtime infer it from the assigned value. After that, though, you're not allowed to store any value of an inappropriate type in that variable.

Dynamic Types in C#

C# 4.0 has features that make it both dynamic and static, as well as both weakly and strongly typed. Though born as a statically typed language, C# becomes dynamically typed in any context in which you use the dynamic keyword, such as this:

```
dynamic number = 10;
Console.WriteLine(number);
```

And because dynamic is a contextual keyword, not a reserved one, this still holds if you have existing variables or methods named dynamic.

Note that C# 4.0 doesn't force you to use dynamic, in the same way that C# 3.0 didn't force you to use var, lambdas or object initializers. C# 4.0 provides the new dynamic keyword specifically to make a few well-known scenarios easier to deal with. The language remains essentially statically typed, even though it has added the ability to interact in a more effective way with dynamic objects.

Why would you want to use a dynamic object? First, you may not know the type of the object you're dealing with. You may have clues but not the certainty to statically type a given variable—which is just what happens in many common situations, such as when you work with COM objects, or when you use reflection to grab instances. In this context, the dynamic keyword makes some situations less painful to deal with. Code written with dynamic is easier to read and write, making for an application that's easier to understand and maintain.

Second, your object may have an inherently changing nature. You may be working with objects created in dynamic programming environments such as IronPython and IronRuby. But you can also use this functionality with HTML DOM objects (subject to expando properties) and the Microsoft .NET Framework 4 objects specifically created to have dynamic natures.

Your best source for software development tools!

programmer's paradise

LEADTOOLS Recognition SDK v16.5

by LEAD Technologies

Develop robust 32/64 bit document imaging and recognition functionality into your applications with accurate and high-speed multi-threaded Forms, OCR, OMR, and 1D/2D barcode engines.

- Supports text, OMR, image, and barcode fields
- Auto-registration and clean-up to improve recognition results
- Provided as both high and low level interface
- Includes comprehensive confidence reports to assess performance

Paradise #
L05 26301A01

$3,214.99

programmers.com/LEAD

Multi-Edit^X

by Multi Edit Software

NEW RELEASE!

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

$198.40

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:

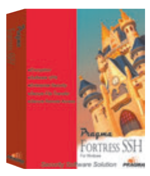
- Increase the productivity of your existing staff three times over
- Control downtime—whether planned or not
- Save more than 50% of the cost of managing, powering and cooling servers

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 Fortress SSH—SSH Server & Client for Windows

by Pragma Systems

Contains SSH, SFTP, SCP servers and clients 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
- Offers FIPS mode
- Hyper-V and PowerShell support
- Runs in Windows 2008R2/2008/2003/7/Vista/XP/2000

Paradise #
P35 04201A01

$550.99

programmers.com/pragma

Camtasia Studio 7

Powerful Screen Recording and Editing for High-quality Videos

by TechSmith

NEW RELEASE!

Delight audiences near or far with training videos, presentations, demos, and screencasts that get results and look fantastic.

Easily record onscreen activity in front of a live audience or from your desk. Capture PowerPoint presentations, webcam video, audio and more. Produce and share high-quality videos at any screen size instantly to YouTube, portable devices and more. Make them wonder how you did it.

Windows
Single User
Paradise #
T08 10501A01

$274.99

programmers.com/techsmith

BUILD ON VMWARE ESXi AND VSPHERE

for Centralized Management, Continuous Application Availability, and Maximum Operational Efficiency in Your Virtualized Datacenter.

Programmer's Paradise invites you to take advantage of this webinar series sponsored by our TechXtend solutions division.

FREE VIRTUALIZATION WEBINAR SERIES: REGISTER TODAY! TechXtend.com/Webinars

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

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

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

$848.99

Download a demo today.

programmers.com/theimagingsource

AdminStudio & Application Virtualization Pack

by Flexera Software

One Application Software Deployment Tool for Reliable MSI Packaging, Application Virtualization, and Windows 7 Migration. Top choice of Microsoft®, Novell®, LANDesk® and other software management solutions. Cut MSI packaging time by up to 70%, Deploy software to desktops with 99% success or better. AdminStudio is the only MSI packaging solution to support multiple virtualization formats, including Microsoft® App-V™, VMware® ThinApp™ and Citrix® XenApp™.

Professional Upgrade from any Active AS Pro + Silver Mtn
Paradise #
I21 09401S05

$4,228.99

programmers.com/flexera

gDoc Fusion

by Global Graphics

gDoc Fusion is the easiest way to assemble information from a range of sources and formats and quickly create enterprise-quality PDF, XPS or Microsoft Word documents for sharing, printing or collaboration.

- One-click PDF/XPS creation
- Merge and combine documents quickly
- Flick through large documents like a book
- Context sensitive text editing

1-4 Users Includes Maint & Support*
Paradise #
F39 ZSP0008

$181.99

* Price per user.

programmers.com/globalgraphics

DON'T BE LEFT BEHIND! STAY ON THE CUTTING EDGE OF TECHNOLOGY:

NEW! MICROSOFT® VISUAL STUDIO® 2010 MAKES IT EASY!

Microsoft® Visual Studio 2010

- Set your ideas free
- Simplicity through integration
- Quality tools help ensure quality results

DISCOVER MICROSOFT VISUAL STUDIO 2010... TAKE YOUR DEVELOPMENT TEAM TO THE NEXT LEVEL!

Call your Programmer's Paradise Representative Today!

866-719-1528 **programmersparadise.com**

Prices subject to change. Not responsible for typographical errors.

Figure 1 **Using dynamic in the Signature of a Function**

```
class Program {
  static void Main(string[] args) {
    // The dynamic variable gets the return
    // value of a function call and outputs it.
    dynamic x = DoubleIt(2);
    Console.WriteLine(x);

    // Stop and wait
    Console.WriteLine("Press any key");
    Console.ReadLine();
  }

  // The function receives and returns a dynamic object
  private static dynamic DoubleIt(dynamic p) {
    // Attempt to "double" the argument whatever
    // that happens to produce

    return p + p;
  }
}
```

Using dynamic

It's important to understand the concept that in the C# type system, dynamic is a type. It has a very special meaning, but it's definitely a type and it's important to treat it as such. You can indicate dynamic as the type of a variable you declare, the type of items in a collection or the return value of a method. You can also use dynamic as the type of a method parameter. Conversely, you can't use dynamic with the typeof operator and you can't use it as the base type of a class.

The following code shows how to declare a dynamic variable in the body of a method:

```
public void Execute()  {
  dynamic calc = GetCalculator();
  int result = calc.Sum(1, 1);
}
```

If you know enough about the type of the object being returned by the GetCalculator method, you can declare the variable calc of that type, or you can declare the variable as var, letting the compiler figure out the exact details. But using var or an explicit static type would require you to be certain that a method Sum exists on the contract exposed by the type GetCalculator returns. If the method doesn't exist, you get a compiler error.

With dynamic, you delay any decision about the correctness of the expression at execution time. The code compiles and is resolved at run time as long as a method Sum is available on the type stored in the variable calc.

You can also use the keyword to define a property on a class. In doing so, you can decorate the member with any visibility modifier you like, such as public, protected, and even static.

Figure 1 shows the versatility of the dynamic keyword. In the main program I have a dynamic variable instantiated with the return value of a function call. That would be no big deal if it weren't for the fact that the function receives and returns a dynamic object. It's interesting to see what happens when, as in the example, you pass a number, then try to double it within the function.

If you feed in a value of 2 and try this code, you receive a value of 4. If you feed in 2 as a string, you'll get 22 instead. Within the function, the + operator is resolved dynamically based on the run time type of the operands. If you change the type to System.Object, you get a compile error, because the + operator isn't defined on System.Object. The dynamic keyword enables scenarios that weren't possible without it.

dynamic vs. System.Object

Until the .NET Framework 4, having a method return different types according to different conditions was possible only by resorting to a common base class. You've probably solved this problem by resorting to System.Object. A function that returns System.Object makes available to the caller an instance that can be cast to nearly anything. So how is using dynamic better than using System.Object?

In C# 4, the actual type behind the variable that's declared dynamic is resolved at run time, and the compiler simply assumes that the object in a variable declared dynamic just supports any operations. This means you can really write code that calls a method on the object you expect to be there at run time, as illustrated here:

```
dynamic p = GetSomeReturnValue();
p.DoSomething();
```

In C# 4.0, the compiler won't complain about that code. The analogous code using System.Object won't compile and requires some hacks on your own—reflection or adventurous casting—in order to work.

var vs. dynamic

The keywords var and dynamic are only apparently similar. Var indicates that the type of the variable has to be set to the compile-time type of the initializer.

But dynamic means that the type of the variable is the dynamic type as available in C# 4.0. In the end, dynamic and var have quite opposite meanings. Var is about reinforcing and improving static typing. It aims to ensure that the type of a variable is inferred by the compiler looking at the exact type being returned by the initializer.

The keyword dynamic is about avoiding static typing altogether. When used in a variable declaration, dynamic instructs the compiler to stop working out the type of the variable at all. The type has to be intended as the type it happens to have at run time. With var, your code is as statically typed as it would have been had you opted for the classic approach of using explicit types in a variable declaration.

Another difference between the two keywords is that var can only appear within a local variable declaration. You can't use var to define a property on a class, nor can you use it to specify the return value or a parameter of a function.

As a developer, you use the dynamic keyword with variables expected to contain objects of uncertain type such as objects returned from a COM or DOM API; obtained from a dynamic language (IronRuby, for example); from reflection; from objects built dynamically in C# 4.0 using the new expand capabilities.

The dynamic type doesn't bypass type checks, though. It only moves them all to run time. If type incompatibilities are discovered at run time, then exceptions are thrown. ■

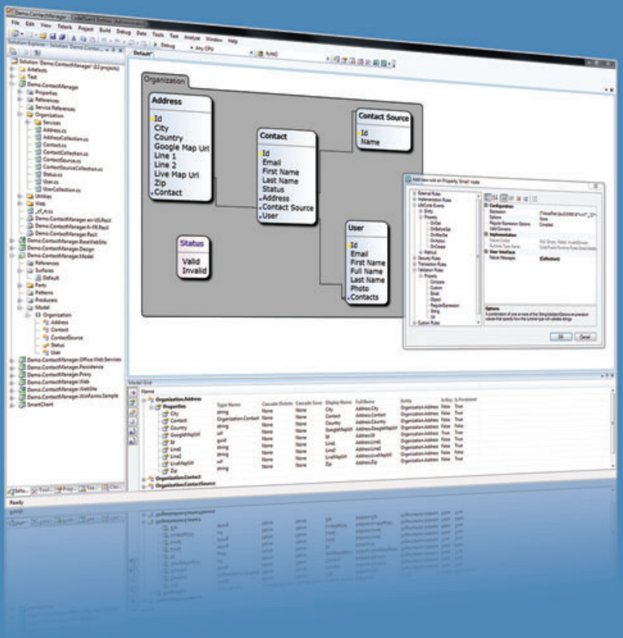
DINO ESPOSITO *is the author of the upcoming "Programming ASP.NET MVC" from Microsoft Press and coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Esposito, who is based in Italy, is a frequent speaker at industry events worldwide. You can join his blog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

THANKS to the following technical expert for reviewing this article: *Eric Lippert*

LOOK AT THAT! IT WORKS ON ALL LAYERS! SL4, WCF SQL, ORACLE, ASP.NET, Winforms, WPF, SharePoint, Office, can that really be true?
WOW! WE'RE READY FOR THE BIG SHOW TOMORROW!
HAHA! 2 YEARS OF WORK!
YOU ARE THE BEST
OH, B.T.W., WE ALSO NEED THE ENTITIES TO BE LOCALIZABLE... ER... WE JUST DECIDED THAT... WELL... YESTERDAY...
CRR..
... ON ALL LAYERS!
YOU ARE THE BEST!

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

RadControls *for* Silverlight

Presenting the industry leading UI components for Silverlight with unmatched performance and pioneering support for Silverlight 4.

Pioneering Support for Microsoft Silverlight 4
Telerik is the first component vendor to provide native controls built on Silverlight 4. RadControls for Silverlight 4 fully match the feature set of their Silverlight 3 counterparts, and closely follow Microsoft's latest advancements in the Silverlight 4 framework, staying up to date with the latest beta releases and the fast-paced Microsoft release schedule for this technology.

Engineered for Great Performance
Telerik Silverlight controls are engineered for outstanding performance by utilizing various techniques that help reduce page loading time and speed up data operations such as streamlined themes and templates, virtualized scrolling, innovative LINQ-based data engine, built-in support for asynchronous databinding, RadCompression module and more.

Support for Visual Studio 2010 and Expression Blend
RadControls for Silverlight provide support for Visual Studio 2010 Beta 2, offering toolbox support, property browsing and WYSIWYG preview for all controls. Telerik is working closely with Microsoft to ensure best practices are followed and provide the most complete design experience, allowing for easy development in Visual Studio 2010 and styling in Expression Blend.

A Comprehensive Silverlight Toolset from the Masters of Web UI
An established leader in web interface technologies, Telerik offers a comprehensive suite of 40+ controls that bring style and interactivity to your line-of-business applications. Featuring a lightning fast DataGrid, rich data visualization controls, and a powerful Outlook-like Scheduling control, RadControls provides all the building blocks for developing next generation Rich Internet Applications (RIAs).

Code Re-Use with RadControls for WPF
RadControls for Silverlight and RadControls for WPF are derived from the same codebase and share the same API. They represent two almost mirror toolsets for building rich line-of-business web and desktop applications, allowing for substantial code and skills reuse between Silverlight and WPF development and shortening your learning curve.

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

FULL
Visual Studio 2010
support

UI COMPONENTS

ASP.NET AJAX
Silverlight
ASP.NET MVC
WinForms
WPF

DATA

OpenAccess ORM
Reporting

PRODUCTIVITY TOOLS

JustCode
JustMock

AUTOMATED TESTING

WebUI Test Studio Developer Edition

TFS TOOLS

Work Item Manager
Project Dashboard

www.telerik.com

telerik
deliver more than expected

JON LANGDON

Production Diagnostics Improvements in CLR 4

On the Common Language Runtime (CLR) team, we have a group whose focus is providing APIs and services that enable others to build diagnostics tools for managed code. The two biggest components we own (in terms of engineering resources dedicated) are the managed debugging and profiling APIs (ICorDebug* and ICorProfiler*, respectively).

Like the rest of the CLR and framework teams, our value is realized only through the applications built on top of our contributions. For example, Visual Studio teams consume these debugging and profiling APIs for their managed debugger and performance profiling tools, and a number of third-party developers are building tools with the profiling API.

For the past 10 years, much of the focus in this area, for both the CLR and Visual Studio, has been on enabling developer desktop scenarios: source-stepping in a debugger to find code errors; launching an application under a performance profiler to help pinpoint slow code paths; edit-and-continue to help reduce time spent in the edit-build-debug cycle; and so on. These tools can be useful for finding bugs in your applications after they've been installed on a user's machine or deployed to a server (both cases hereafter referred to as *production*), and we do have a number of third-party vendors building world-class production diagnostics tools on top of our work.

Bugs are more costly to fix the later they're found in the application lifecycle.

However, we consistently get feedback from customers and these vendors stressing the importance of making it even easier to find bugs throughout the life of an application. After all, software bugs are generally considered to be more costly to fix the later they're found in the application lifecycle.

CLR 4 (the runtime underlying the Microsoft .NET Framework 4) is the first release in which we've made a significant effort to address that feedback and to begin expanding the scenarios our diagnostic APIs support toward the production end of the spectrum.

In this article, I will take a look at some of the scenarios we understand to be particularly painful today, the approach we're taking to solve them and the types of tools they enable. Specifically, I will explain how we've evolved the debugging API to support dump debugging for application-crash and -hang scenarios, and how we've made it easier to detect when hangs are caused by multi-threading issues.

I will also describe how adding the ability to attach profiling tools to an already-running application will further ease troubleshooting these same scenarios, and greatly reduce the amount of time it takes to diagnose problems caused by excessive memory consumption.

Finally, I will briefly explain how we made profiling tools easier to deploy by removing the dependency on the registry. Throughout, the focus is primarily on the types of new tools our work enables, but where applicable I have included references to additional resources that will help you understand how you can take advantage of our work through Visual Studio.

Dump Debugging

One popular feature we're delivering with Visual Studio 2010 is managed dump debugging. Process dumps, typically referred to just as dumps, are commonly used in production debugging scenarios for both native and managed code. A dump is essentially a snapshot of a process' state at a given point in time. Specifically, it's the contents of the process's virtual memory (or some subset thereof) dumped into a file.

Prior to Visual Studio 2010, in order to debug managed code in dumps you needed to use the specialized Windows Debugger extension sos.dll to analyze the dumps, instead of more familiar tools like Visual Studio (where you probably wrote and debugged your code during product development). Our goal for the high-level experience we want you to have when using a dump to diagnose issues in Visual Studio is that of stopped-state live debugging: what you experience when you're debugging code and stopped at a breakpoint.

The most common point in time to collect a dump is when there's an unhandled exception in an application—a crash. You use the dump to figure out why the crash occurred, typically starting by looking at the call stack of the faulting thread. Other scenarios where dumps are used are application hangs and memory-usage issues.

This article discusses prerelease versions of the Microsoft .NET Framework 4 and Visual Studio 2010. All information is subject to change.

Post your questions and comments on the CLR Team blog at blogs.msdn.com/clrteam.

Thanks for Making DevExpress #1

WinForms Controls — Win

ASP.NET AJAX Controls — ASP

Silverlight Controls — Ag

WPF Controls — WPF

Visual Studio Tools — IDE

Application Frameworks

Free Trial Version: DevExpress.com/eval

AWARD WINNING DEVELOPMENT TOOLS

Feature-Complete Presentation Components • Easy-to-Use Reporting Controls
IDE Productivity Tools • Business Application Frameworks

DevExpress™
Download • Compare • Decide!™

For example, if your Web site has stopped processing requests, you might attach a debugger, gather a dump and restart the application. Offline analysis of the dump might show, for example, that all of your threads processing requests are waiting on a connection to the database, or perhaps you find a deadlock in your code. Memory-usage issues can manifest in various ways from an end user's perspective: the application slows down because of excessive garbage collection; service is interrupted because the application ran out of virtual memory and needed to be restarted; and so forth.

Through CLR 2, the debugging APIs provided support for debugging running processes only, which made it difficult for tools to target the scenarios just described. Basically, the API was not designed with dump debugging scenarios in mind. The fact that the API uses a helper thread running in the target process to service debugger requests underscores this point.

For example, in CLR 2, when a managed debugger wants to walk a thread's stack, it sends a request to the helper thread in the process being debugged. The CLR in that process services the request and returns the results to the debugger. Because a dump is just a file, there's no helper thread to service requests in this case.

A dump is essentially a snapshot of a process' state at a given point in time.

To provide a solution for debugging managed code in a dump file, we needed to build an API that did not require running code in the target to inspect managed-code state. Yet because debugger writers (primarily Visual Studio) already have a significant investment in the CLR debugging API to provide live debugging, we did not want to force the use of two different APIs.

Where we landed in CLR 4 was reimplementing a number of the debugger APIs (primarily those required for code and data inspection) to remove the use of the helper thread. The result is that the existing API no longer needs to care whether the target is a dump file or a live process. In addition, debugger writers are able to use the same API to target both live and dump debugging scenarios. When live debugging specifically for execution control—setting breakpoints and stepping through code—the debugging API still uses a helper thread. Over the long term, we intend to remove the dependency for these scenarios as well. Rick Byers (a former developer on the debugging services API) has a useful blog post describing this work in more detail at blogs.msdn.com/rmbyers/archive/2008/10/27/icordebug-re-architecture-in-clr-4-0.aspx.

You can now use ICorDebug to inspect managed code and data in a dump file: walk stacks, enumerate locals, get the exception type and so on. For crashes and hangs, there's often enough context available from the thread stacks and ancillary data to find the cause of the issue.

While we know the memory diagnostics and other scenarios are also important, we simply did not have enough time in the CLR 4 schedule to build new APIs that give debuggers the ability to inspect the managed heap in the way this scenario requires. Moving forward, as we continue to expand the production diagnostics scenarios we support, I expect this is something we will add. Later in the article I discuss other work we've done to help address that scenario.

I'd also like to explicitly call out that this work supports both 32- and 64-bit targets and both managed-only and mixed-mode (native and managed) debugging. Visual Studio 2010 provides mixed-mode for dumps containing managed code.

Monitor Lock Inspection

Multi-threaded programming can be difficult. Whether you are explicitly writing multi-threaded code or leveraging frameworks or libraries that are doing it for you, diagnosing issues in asynchronous and parallel code can be quite challenging. When you have a logical unit of work executing on a single thread, understanding causality is much more straightforward and can often be determined by simply looking at the thread's call stack. But when that work is divided among multiple threads, tracing the flow becomes much harder. Why is the work not completing? Is some portion of it blocked on something?

With multiple cores becoming commonplace, developers are looking more and more to parallel programming as a means for performance improvement, rather than simply relying on chip speed advancements. Microsoft developers are included in this lot, and over the past few years we've been significantly focused on making it easier for developers to be successful in this area. From a diagnostics perspective, we've added a few simple-yet-helpful APIs that enable tools to help developers better cope with the complexities of multi-threaded code.

To the CLR debugger APIs we've added inspection APIs for monitor locks. Simply put, monitors provide a way for programs to synchronize access to a shared resource (some object in your .NET code) across multiple threads. So while one thread has the resource locked, another thread waits for it. When the thread owning the lock releases it, the first thread waiting may now acquire the resource.

In the .NET Framework, monitors are exposed directly through the System.Threading.Monitor namespace, but more commonly through the lock and SyncLock keywords in C# and Visual Basic, respectively. They're also used in the implementation of synchronized methods, the Task Parallel Library (TPL) and other asynchronous programming models. The new debugger APIs let you better understand what object, if any, a given thread is blocked on and what thread, if any, holds a lock on a given object. Leveraging these APIs, debuggers can help developers pinpoint deadlocks and understand when multiple threads contending for a resource (lock convoys) may be affecting an application's performance.

For an example of the type of tools this work enables, check out the parallel debugging features in Visual Studio 2010. Daniel Moth and Stephen Toub provided a great overview of these in the September 2009 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee410778).

One of the things that excites us the most about the dump debugging work is that building an abstracted view of the debug target means

"We were previously using a rival suite of components – but your components are by far better than anything else I have seen on the market. I can't believe I haven't come across your components before. Using the DevExpress suite has saved so much time, allowing us to focus on the core purpose of our system."

"There were 3 main factors that have led me to choose CodeRush & Refactor! Pro as my main productivity tool. 1. Great ROI 2. Short Learning Curve 3. Great Support."

"I'm pleasantly surprised at the quality, performance and lightweight nature of the DevExpress components. I would like to congratulate the team on a great job."

"What can I say about CodeRush and Refactor! Pro other than, "WOW!"

"I've been using CodeRush Xpress since a little after it was first announced, and every new release has gotten better and better at filling the gaping hole that is refactoring and coding assistance within Visual Studio."

"We've done a lot of research comparing the various ASP.NET controls. All the forums and comments we've read said that DevExpress is the way to go. I haven't read one negative thing about them and that says a lot! We're looking forward to refreshing our existing applications and new ones with the jaw dropping features of DevExpress."

"Your product line is outstanding and has taken our product lines to a new level. It is very rare to get exactly what you think you're buying, never mind your tradition of delivering more and better than expected."

download
compare
decide!

www.DevExpress.com

WinForms Controls	ASP.NET AJAX Controls	Silverlight Controls	WPF Controls	Visual Studio Tools	Application Frameworks
Win	ASP	Ag	WPF	IDE	

Free Trial Version: DevExpress.com/eval

AWARD WINNING DEVELOPMENT TOOLS

Feature-Complete Presentation Components • Easy-to-Use Reporting Controls
IDE Productivity Tools • Business Application Frameworks

DevExpress™
Download • Compare • Decide!™

new inspection functionality is added, such as the monitor-lock-inspection feature, which provides value for both live and dump debugging scenarios. While I expect this feature to be extremely valuable for developers while they're initially developing an application, it's the dump debugging support that makes monitor-lock inspection a compelling addition to the production diagnostic features in CLR 4.

Tess Ferrandez, a Microsoft support engineer, has a Channel 9 video (channel9.msdn.com/posts/Glucose/Hanselminutes-on-9-Debugging-Crash-Dumps-with-Tess-Ferrandez-and-VS2010/) in which she simulates a lock-convoy scenario common to what she has found when troubleshooting customer applications. She then walks through how to use Visual Studio 2010 to diagnose the problem. It's a great example of the types of scenarios these new features enable.

Beyond Dumps

While we believe the tools these features enable will help decrease the amount of time it takes developers to resolve issues in production, we do not expect (nor want) dump debugging to be the only way to diagnose production issues.

In the case of diagnosing issues of excessive memory use issues, we usually start with a list of object instances grouped by type with their count and aggregate size and progress toward understanding object reference chains. Shuttling dump files containing this information between production and development machines, operations staff, support engineers and developers can be unwieldy and time-consuming. And as applications grow in size—this is especially prevalent with 64-bit applications—the dump files also grow in size and take longer to move around and process. It was with these high-level scenarios in mind that we undertook the profiling features described in the following sections.

Profiling Tools

There are a number of different types of tools built on top of the CLR's profiling API. Scenarios involving the profiling API generally focus on three functional categories: performance, memory and instrumentation.

Performance profilers (like the one that ships in some of the Visual Studio versions) focus on telling you where your code is spending time. Memory profilers focus on detailing your application's memory consumption. Instrumenting profilers do, well, everything else.

Let me clarify that last statement a bit. One of the facilities the profiling API provides is the ability to insert intermediate language (IL) into managed code at run time. We call this code instrumentation. Customers use this functionality to build tools that deliver a wide range of scenarios from code coverage to fault injection to enterprise-class production monitoring of .NET Framework-based applications.

One of the benefits the profiling API has over the debugging API is that it is designed to be extremely lightweight. Both are event-driven APIs—for example, there are events in both for assembly load, thread create, exception thrown and so on—but with the profiling API you register only for the events you care about. Additionally, the profiling DLL is loaded inside the target process, ensuring fast access to run time state.

In contrast, the debugger API reports every event to an out-of-process debugger, when attached, and suspends the runtime on each event. These are just a couple of the reasons the profiling API is an attractive option for building online diagnostics tools targeting production environments.

Profiler Attach and Detach

While several vendors are building always-on, production-application-monitoring tools via IL instrumentation, we do not have many tools leveraging the performance- and memory-monitoring facilities in the profiling API to provide reactive diagnostics support. The main impediment in this scenario has been the inability to attach CLR profiling API-based tools to an already-running process.

In versions preceding CLR 4, the CLR checks during startup to see if a profiler was registered. If it finds a registered profiler, the CLR loads it and delivers callbacks as requested. The DLL is never unloaded. This is generally fine if the tool's job is to build an end-to-end picture of application behavior, but it does not work for problems you didn't know existed when the application started.

Perhaps the most painful example of this is the memory-usage-diagnostics scenario. Today, in this scenario we often find that the pattern of diagnostics is to gather multiple dumps, look at the differences in allocated types between each, build a timeline of growth, and then find in the application code where the suspicious types are being referenced. Perhaps the issue is a poorly implemented caching scheme, or maybe event handlers in one type are holding references to another type that's otherwise out of scope. Customers and support engineers spend a lot of time diagnosing these types of issues.

For starters, as I briefly mentioned earlier, dumps of large processes are large themselves, and getting them to an expert for diagnosis can introduce long delays in the time to resolution. Further, there is the problem that you have to involve an expert, primarily due to the fact that the data is only exposed via a Windows Debugger extension. There's no public API that allows tools to consume the data and build intuitive views on top of it or integrate it with other tools that could aid in analysis.

Multi-threaded programming can be difficult.

To ease the pain of this scenario (and some others), we added a new API that allows profilers to attach to a running process and leverage a subset of the existing profiling APIs. Those available after attaching to the process allow sampling (see "VS 2010: Attaching the Profiler to a Managed Application" at blogs.msdn.com/profiler/archive/2009/12/07/vs2010-attaching-the-profiler-to-a-managed-application.aspx) and memory diagnostics: walking the stack; mapping function addresses to symbolic names; most of the Garbage Collector (GC) callbacks; and object inspection.

In the scenario I just described, tools can leverage this functionality to allow customers to attach to an application experiencing slow response times or excessive memory growth, understand what's

Read more about
our awards and user comments

DevExpress.com/awards
DevExpress.com/comments

#1 PRODUCT
ComponentSource® Awards 2009-10

#1 PUBLISHER
ComponentSource® Awards 2009-10

Readers' Choice 2009 asp.netPRO Winner

Visual Studio Magazine Readers Choice 2009

WindowsITPro | SQL Server — Microsoft tech·ed North America 2009 Attendees' Pick

WindowsITPro | SQL Server — Best of Microsoft tech·ed North America 2009 Winner

WinForms Controls	ASP.NET AJAX Controls	Silverlight Controls	WPF Controls	Visual Studio Tools	Application Frameworks
Win	ASP	Ag	WPF	IDE	

Free Trial Version: DevExpress.com/eval

AWARD WINNING DEVELOPMENT TOOLS

Feature-Complete Presentation Components • Easy-to-Use Reporting Controls
IDE Productivity Tools • Business Application Frameworks

DevExpress™
Download • Compare • Decide!™

currently executing, and know what types are alive on the managed heap and what's keeping them alive. After gathering the information, you can detach the tool and the CLR will unload the profiler DLL. While the rest of the workflow will be similar—finding where these types are referenced or created in your code—we expect tools of this nature to have a sizeable impact on the mean-time-to-resolution for these issues. Dave Broman, developer on the CLR profiling API, discusses this and other profiling features in more depth on his blog (blogs.msdn.com/davbr).

I do, however, want to call out two limitations explicitly in this article. First, memory-diagnostics on attach is limited to non-concurrent (or blocking) GC modes: when the GC suspends execution of all managed code while performing a GC. While concurrent GC is the default, ASP.NET uses server-mode GC, which is a non-concurrent mode. This is the environment in which we see most of these issues. We do appreciate the usefulness of attach-profiling diagnostics for client applications and expect to deliver this in a future release. We simply prioritized the more common case for CLR 4.

Second, you are unable to use the profiling API to instrument IL after attach. This functionality is especially important to our enterprise-monitoring tool vendors who want to be able to change their instrumentation dynamically in response to runtime conditions. We commonly refer to this feature as re-JIT. Today, you have one opportunity to change the IL body of a method: when it's first being JIT-compiled. We expect delivering re-JIT to be a significant and important undertaking and are actively investigating both the customer value and technical implications as we consider delivering the work in a future release.

Registry-Free Activation for Profilers

Profiler attach and the supporting work of enabling specific profiling APIs after attach was the biggest piece of work we did in the CLR profiling API for CLR 4. But we were delighted to find another very small (in terms of engineering cost) feature that has a surprisingly large impact on customers building production-class tools.

Memory profilers focus on detailing your application's memory consumption.

Prior to CLR 4, for a tool to get its profiler DLL loaded in a managed application it had to create two key pieces of configuration information. The first was a pair of environment variables that told the CLR to enable profiling and what profiler implementation to load (its CLSID or ProgID) when the CLR starts up. Given that profiler DLLs are implemented as in-process COM servers (with the CLR being the client), the second piece of configuration data was the corresponding COM registration information stored in the registry. This basically told the runtime, by way of COM, where to find the DLL on disk.

During CLR 4 planning, while trying to understand how we could make it easier for vendors to build production-class tools, we had some interesting conversations with some of our support engineers and tools vendors. The feedback we received was that, when faced with an application failure in production, customers often care less about impact to the application (it's already failing; they just want to get the diagnostic information and move on) and more about impact to the state of the machine. Many customers go to great lengths to document and police the configuration of their machines, and introducing changes to that configuration can introduce risk.

So, with respect to the CLR's part of the solution, we wanted to make it possible to enable xcopy-deployable diagnostics tools to both minimize the risk of state changes and to reduce the time it takes to get the tools up and running on the machine.

In CLR 4 we removed the need to register the COM profiler DLL in the registry. We still require environment variables if you want to launch your application under the profiler, but in the attach scenarios detailed previously, there is no configuration required. A tool simply calls the new attach API, passes the path to the DLL and the CLR loads the profiler. Moving forward, we're looking into ways to further simplify profiler configuration as customers still find the environment-variable solution challenging in some scenarios.

Combined, the two profiling features I just discussed enable a class of low-overhead tools that customers can, in the case of unexpected failures, quickly deploy to a production machine to gather performance or memory diagnostic information to help pinpoint the cause of the problem. Then, just as quickly, the user can return the machine to its original state.

Wrapping Up

Working on diagnostics features for the CLR is bittersweet. I see many examples of how much customers struggle with certain issues, and yet there's always cool work we can do to help ease development and make customers happier. If that list of issues challenging our customers isn't changing—that is, we're not removing items from it—then we're not succeeding.

In CLR 4 I think we built features that not only provide immediate value in addressing some top customer problems, but also lay a foundation upon which we can continue to build in the future. Moving forward, we will continue to invest in APIs and services that make it even easier to expose meaningful diagnostic information in all phases of an application's life so you can focus more time on building new features for your customers and less time debugging existing ones.

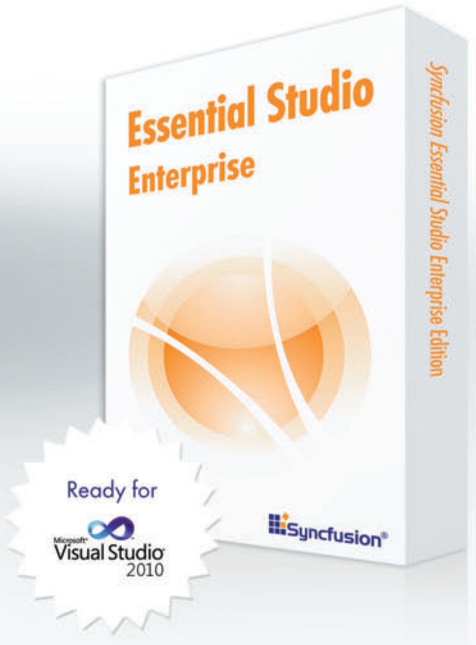
If you're interested in providing us feedback on diagnostics work we're considering for our next release, we've posted a survey at surveymonkey.com/s/developer-productivity-survey. As we are now about to ship CLR 4, we're focusing much of our time on planning for upcoming releases and data from the survey will help us prioritize our efforts. If you have a few minutes, we would love to hear from you. ■

JON LANGDON *is a program manager on the CLR team where he focuses on diagnostics. Prior to joining the CLR team, he was a consultant with Microsoft Services, helping customers diagnose and fix problems with large-scale, enterprise applications.*

THANKS to the following technical expert for reviewing this article: *Rick Byers*

Deliver innovation with ease®

with Syncfusion's award-winning .NET components and controls


Essential Studio
Enterprise
Syncfusion Essential Studio Enterprise Edition
Syncfusion®
Ready for
Microsoft Visual Studio 2010

User interfaces, reporting, business intelligence.
Any .NET platform. Just one package.

ASP.NET ASP.NET MVC Windows Forms WPF Silverlight

Syncfusion®

www.syncfusion.com 1 888-9 DOTNET

PROGRAMMING LANGUAGES

Embarcadero Technologies
Embarcadero empowers software developers and database professionals with award-winning tools including Delphi Prism for rapidly developing .NET and ASP.NET applications.

CODING & BUILD TOOLS

AppDev
Award-winning, practical learning solutions for Microsoft developers integrated into Visual Studio 2010. Courses for Visual Studio, SQL, SharePoint and more.

ArtinSoft
ArtinSoft's unique tools and services allow cost-effectively converting Visual Basic 6.0 applications to Visual Studio 2010 and .NET Framework 4

Flexera
InstallShield makes it easy to build professional installations for desktop, server, Web, and mobile applications developed in Visual Studio 2010.

GraFX
Vulcan.NET is the next generation of the xBase family of languages for the Microsoft .NET Platform, opening up new opportunities for xBase (FoxPro, Clipper and Visual Objects) programmers.

APPLICATION LIFECYCLE M

GamCom
GamCom provides software and consultancy services. Talmia is an innovative ALM process automation and reporting tool for TFS.

/*SHOWCASE YOUR INNO

Find extensions on the Visual Studio Gallery
www.visualstudiogallery.com

Join the Wave of Innovation
http://msdn.com/vsip

COMPONENTS & LIBRARIES

DevExpress
DevExpress engineers feature-complete Presentation Components, Reporting Systems, IDE Productivity Tools and Business Application Frameworks for Visual Studio .NET.

GrapeCity
GrapeCity provides complete Spreadsheet, Reporting, Analysis and Business Intelligence Tools for .NET, including the ActiveReports and FarPoint Spread product lines.

ComponentOne
ComponentOne, a leading component vendor, provides custom WinForms, WPF, ASP.NET AJAX, Silverlight, Mobile, and COM components, documentation tools, and Web Parts.

APPLICATION MIGRATION & MODERNIZATION

Alchemy
Provides modernize-as-you-migrate options for COBOL applications. NetCOBOL for .NET brings the latest Visual Studio 2010 productivity features to COBOL programmers.

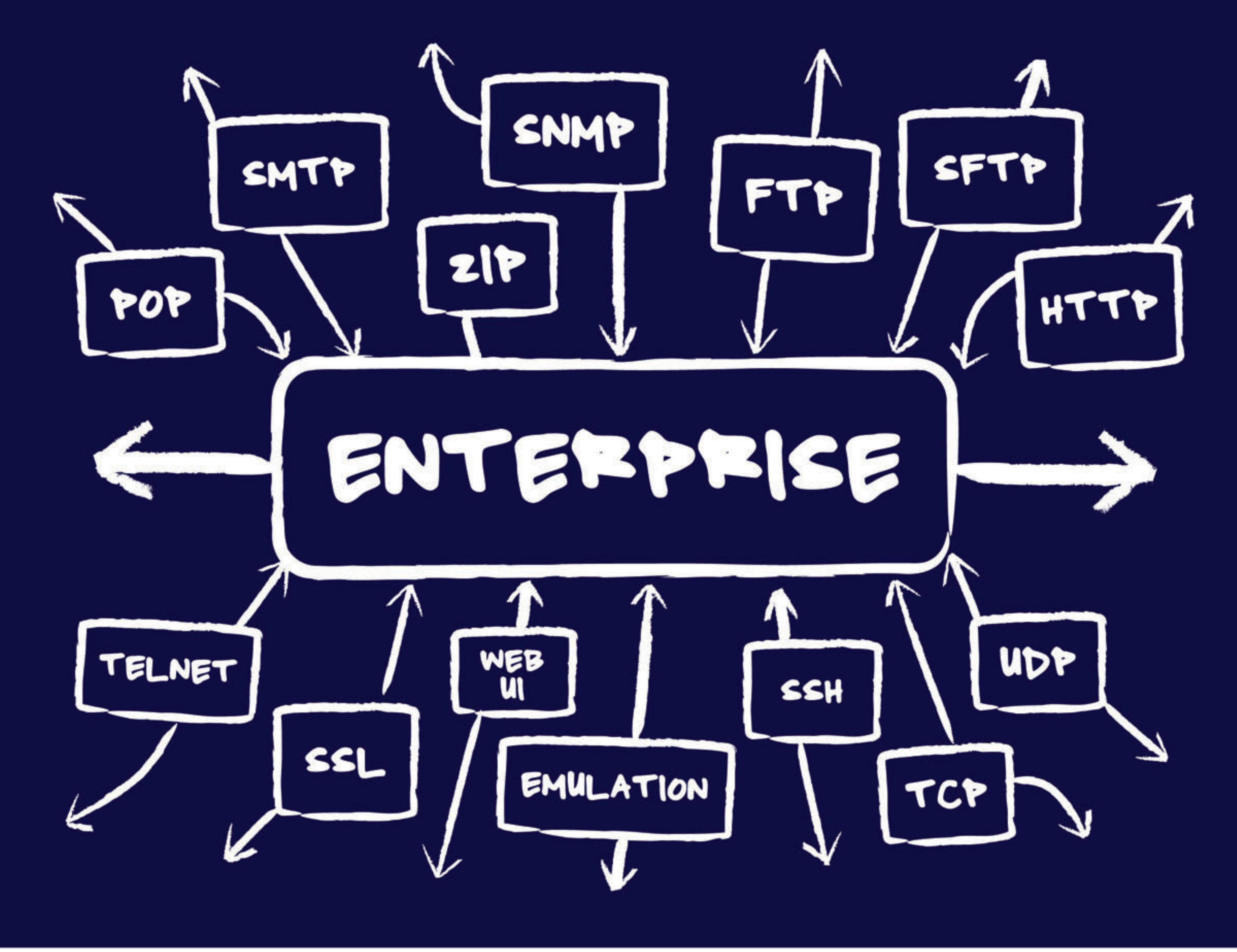
MANAGEMENT TOOLS

dynaTrace
Eliminate manual problem reproduction, rapidly isolate and diagnose .NET performance issues to code-level with dynaTrace's extension for VS Ultimate 2010

VATION*/

LOOK WHAT WE DID WITH VISUAL STUDIO 2010!

Microsoft® Visual Studio®



Internet Connectivity for the Enterprise

Since 1994, Dart has been a leading provider of high quality, high performance Internet connectivity components supporting a wide range of protocols and platforms. Dart's three product lines offer a comprehensive set of tools for the professional software developer.

PowerSNMP for ActiveX and .NET

Create custom Manager, Agent and Trap applications with a set of native ActiveX, .NET and Compact Framework components. **SNMPv1**, **SNMPv2**, **SNMPv3** (authentication/encryption) and **ASN.1** standards supported.

PowerWEB for ASP.NET

AJAX enhanced user interface controls for responsive ASP.NET applications. Develop unique solutions by including streaming file upload and interactive image pan/zoom functionality within a page.

PowerTCP for ActiveX and .NET

Add high performance Internet connectivity to your ActiveX, .NET and Compact Framework projects. Reduce integration costs with detailed documentation, hundreds of samples and an expert in-house support staff.

SSH	**FTP**	**SMTP**	**DNS**	**Telnet**
UDP	**SFTP**	**IMAP**	**Rlogin**	**VT Emulation**
TCP	**HTTP**	**S/MIME**	**Rsh**	**ZIP Compression**
SSL	**POP**	**Ping**	**Rexec**	*more...*

Ask us about Mono Platform support. Contact sales@dart.com.

Download a fully functional product trial today!

DART.com

LINQ Projection Queries and Alternatives in WCF Services

The presenter at my local .NET user group was writing a LINQ query during his session last month when I asked him, "How did we ever live without LINQ?" "I have no idea," he replied.

It's true. Since it was introduced in Visual Studio 2008, LINQ has made *such* a difference in how we code in the Microsoft .NET Framework. In combination with the many new language features that were introduced in Visual Basic and C#, it's a consistent problem solver for querying in-memory objects and data sources.

One of LINQ's abilities that is both a blessing and an occasional source of frustration is that it can project randomly shaped data into anonymous types. When you simply need to grab a special view of your data, without having to declare a new class for this throwaway type, anonymous types are a great solution. LINQ projections and anonymous types have certainly spoiled us. So why do I say they can also be a source of frustration?

If you have ever used a LINQ projection in a method that needs to return data to another method—or worse, used a LINQ projection in a Windows Communication Foundation (WCF) service operation—you may understand.

Because anonymous types are throwaway types, they have no declaration and are understood only within the method where they're created. If you write a query that returns a list of anonymous types, there's no way to define a method argument to say "I'm going to return a list of … " because there's no way to express "… of anonymous types."

Here's a LINQ to Entities query with a simple projection:

```
var custQuery = from c in context.Customers
                select new {c.CustomerID, Name=c.LastName.Trim() +
                ", " + c.FirstName};
```

At run time, the custQuery variable will actually be an ObjectQuery<<>f__AnonymousType0<int,string>>.

The var (and the alternate use of Visual Basic Dim) allows us to get away with not having (or needing) a way to express this non-type.

If you want to return the results of that query from a method, the only reasonable solution is to create a class to represent the type being returned. Doing this, however, renders the beauty of the anonymous type moot. Now you have to write more code, define classes and (possibly) new projects to house the new classes, ensure the various assemblies using these classes have access to them and so on.

Until recently, data services provided an additional conundrum. In order to project data, you had to create a custom operation in a service, execute your own query and then return some type of pre-defined class that could be understood by the client.

When you're working with services, there are many scenarios where you want to work with a particular view of data without paying the price of moving larger types across the wire.

It turns out, there are more options besides creating an extra type in your domain to satisfy this temporary need.

New Projection Capability in WCF Data Services

The Data Services Update for the .NET Framework 3.5 SP1 introduces a handful of powerful features for WCF Data Services, which are also part of the .NET Framework 4. Among these features is the ability to use projections in queries against the data services. I highly recommend checking out the WCF Data Services team blog post on all that's new in this update at blogs.msdn.com/astoriateam/archive/2010/01/27/data-services-update-for-net-3-5-sp1-available-for-download.aspx.

The $select operator has been added to the data services URI syntax. It allows for property and even navigation property projection.

Here's a simple example of a projection that gets a few scalar properties for a customer along with the SalesOrderHeaders navigation property:

```
http://localhost /DataService.svc/Customers(609)
  ?$select=CustomerID,LastName,FirstName,SalesOrderHeaders&$expand=
  SalesOrderHeaders
```

The expand operator forces the results to include not just a link to those orders, but the data for each order as well.

Figure 1 shows the results of this query. The expanded SalesOrderHeaders (which contains only a single order) is highlighted in yellow while the customer information is highlighted in green.

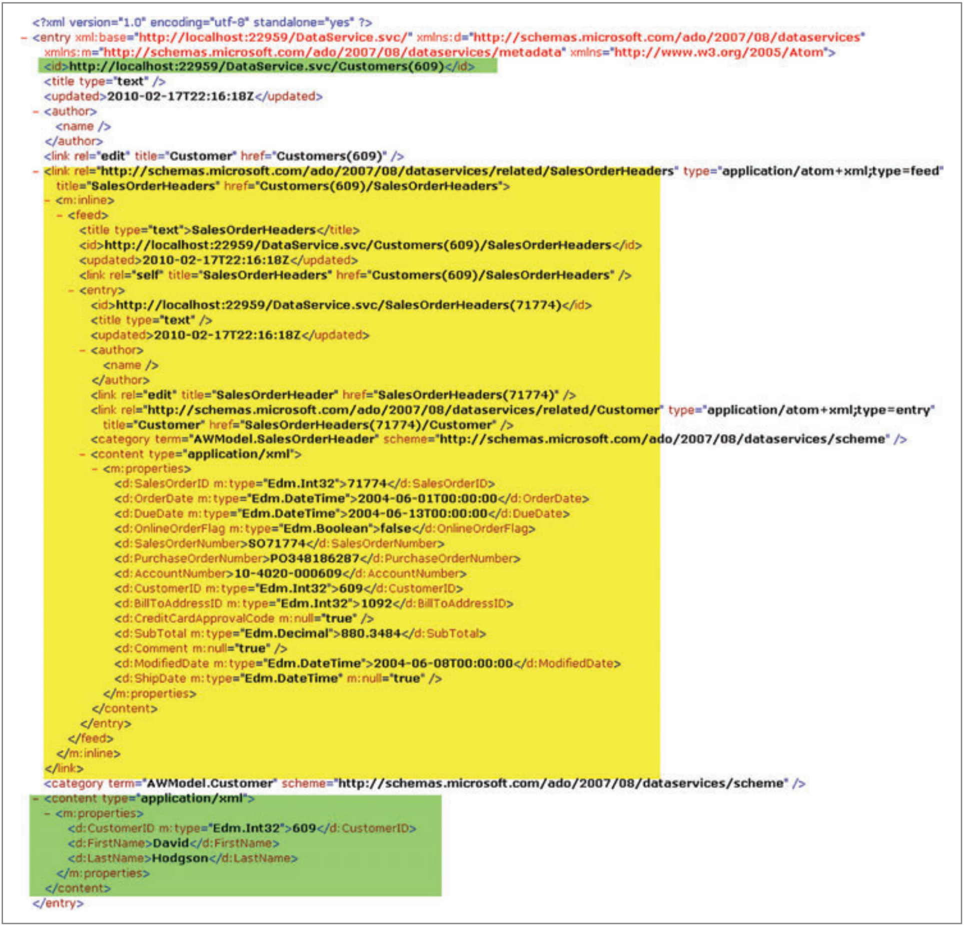
The LINQ to REST feature in the .NET Framework and Silverlight client APIs for WCF Data Services has been updated to allow projections as well:

```
var projectedCust = (from c in context.Customers
                     where c.CustomerID==609
                     select new {c.CustomerID, c.LastName})
                     .FirstOrDefault();
```

ProjectedCust is now an anonymous type I can use in my client application.

It's also possible to project into known entity types, and in some cases, the DataContext can keep track of changes made by the client and these changes can be persisted back through the service's SaveChanges method. Be aware that any missing properties will get populated with their defaults (or null if they're nullable) and be persisted to the database.

Code download available at code.msdn.microsoft.com/mag201005DataPoints.



```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <entry xml:base="http://localhost:22959/DataService.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">
   <id>http://localhost:22959/DataService.svc/Customers(609)</id>
   <title type="text" />
   <updated>2010-02-17T22:16:18Z</updated>
 - <author>
     <name />
   </author>
   <link rel="edit" title="Customer" href="Customers(609)" />
 - <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/SalesOrderHeaders" type="application/atom+xml;type=feed"
     title="SalesOrderHeaders" href="Customers(609)/SalesOrderHeaders">
   - <m:inline>
     - <feed>
         <title type="text">SalesOrderHeaders</title>
         <id>http://localhost:22959/DataService.svc/Customers(609)/SalesOrderHeaders</id>
         <updated>2010-02-17T22:16:18Z</updated>
         <link rel="self" title="SalesOrderHeaders" href="Customers(609)/SalesOrderHeaders" />
       - <entry>
           <id>http://localhost:22959/DataService.svc/SalesOrderHeaders(71774)</id>
           <title type="text" />
           <updated>2010-02-17T22:16:18Z</updated>
         - <author>
             <name />
           </author>
           <link rel="edit" title="SalesOrderHeader" href="SalesOrderHeaders(71774)" />
           <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer" type="application/atom+xml;type=entry"
             title="Customer" href="SalesOrderHeaders(71774)/Customer" />
           <category term="AWModel.SalesOrderHeader" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
         - <content type="application/xml">
           - <m:properties>
               <d:SalesOrderID m:type="Edm.Int32">71774</d:SalesOrderID>
               <d:OrderDate m:type="Edm.DateTime">2004-06-01T00:00:00</d:OrderDate>
               <d:DueDate m:type="Edm.DateTime">2004-06-13T00:00:00</d:DueDate>
               <d:OnlineOrderFlag m:type="Edm.Boolean">false</d:OnlineOrderFlag>
               <d:SalesOrderNumber>SO71774</d:SalesOrderNumber>
               <d:PurchaseOrderNumber>PO348186287</d:PurchaseOrderNumber>
               <d:AccountNumber>10-4020-000609</d:AccountNumber>
               <d:CustomerID m:type="Edm.Int32">609</d:CustomerID>
               <d:BillToAddressID m:type="Edm.Int32">1092</d:BillToAddressID>
               <d:CreditCardApprovalCode m:null="true" />
               <d:SubTotal m:type="Edm.Decimal">880.3484</d:SubTotal>
               <d:Comment m:null="true" />
               <d:ModifiedDate m:type="Edm.DateTime">2004-06-08T00:00:00</d:ModifiedDate>
               <d:ShipDate m:type="Edm.DateTime" m:null="true" />
             </m:properties>
           </content>
         </entry>
       </feed>
     </m:inline>
   </link>
   <category term="AWModel.Customer" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
 - <content type="application/xml">
   - <m:properties>
       <d:CustomerID m:type="Edm.Int32">609</d:CustomerID>
       <d:FirstName>David</d:FirstName>
       <d:LastName>Hodgson</d:LastName>
     </m:properties>
   </content>
</entry>
```

Figure 1 **Results of a Data Services Query Projection Requesting Three Customer Properties and the Customer's SalesOrderHeaders**

Enabling Projected Strong Types from an EDM

If you're using an Entity Framework Entity Data Model (EDM), there's a convenient way to avoid being stuck with anonymous types when you need to pass them out of the method in which they were created.

The EDM has a mapping called QueryView. I've pointed many clients to this in the past, prior to data services projection support. Not only does it solve the problem nicely for data services, but for custom WCF Services and RIA Services as well.

What is a QueryView? It's a special type of mapping in the Entity Framework metadata. Typically, you map properties of an entity to database tables or view columns as they're described in the store model—Storage Schema Definition Language (SSDL)—of metadata, as shown in **Figure 2**.

A QueryView, however, lets you create a view over those SSDL table columns rather than map directly to them. There are many reasons to use a QueryView. Some examples include: to expose your entities as read-only, to filter entities in a way that conditional mapping does not allow or to provide different views of the data tables from the database.

It's the last of these purposes that I will focus on as an alternative to the anonymous types you frequently find yourself projecting in your application. One example would be a pick list. Why return an entire customer type for a drop-down that needs only an ID and the customer's name?

Building a QueryView

Before creating a QueryView, you need to create an entity in the model that represents the shape of the view you're aiming for—for example, the CustomerNameAndID entity.

But you can't map this entity directly to the Customer table in SSDL. Mapping both the Customer entity and the CustomerNameAndID entity to the table's CustomerID column would create a conflict.

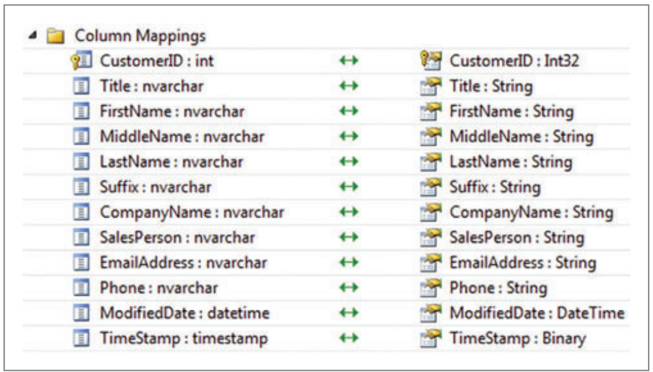
Instead, just as you can create a view of a table in your database, you can create a view of the SSDL Customer directly in the metadata. A QueryView is literally an Entity SQL expression over the SSDL. It's part of the mapping specification language (MSL) metadata of the model. There is no designer support to create the QueryView, so you'll need to type it directly in the XML.

Because you'll be mapping to the store schema of the table, it's a good idea to see what that looks like. **Figure 3** lists the SSDL description of the Customer database table, which looks similar to the Customer entity in the conceptual model's metadata, except for the use of provider data types.

Another important element for the QueryView will be the store schema's namespace, ModelStoreContainer. Now you have the pieces necessary to construct the QueryView expression. Here's a QueryView that projects the three required fields from the SSDL into the CustomerNameAndID entity that I created in the model:

```
SELECT VALUE AWModel.CustomerNameAndID(c.CustomerID, c.FirstName,
  c.LastName) FROM ModelStoreContainer.Customer as c
```

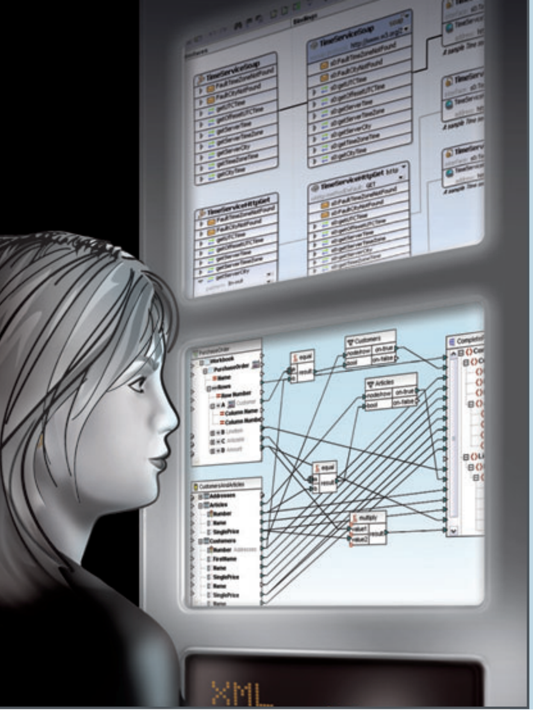
Translating the Entity SQL to English: "Query the Customer in the store schema, pull out these three columns and give them back to me as a CustomerNameAndID entity." AWModel is the



Column Mappings

CustomerID : int	↔	CustomerID : Int32
Title : nvarchar	↔	Title : String
FirstName : nvarchar	↔	FirstName : String
MiddleName : nvarchar	↔	MiddleName : String
LastName : nvarchar	↔	LastName : String
Suffix : nvarchar	↔	Suffix : String
CompanyName : nvarchar	↔	CompanyName : String
SalesPerson : nvarchar	↔	SalesPerson : String
EmailAddress : nvarchar	↔	EmailAddress : String
Phone : nvarchar	↔	Phone : String
ModifiedDate : datetime	↔	ModifiedDate : DateTime
TimeStamp : timestamp	↔	TimeStamp : Binary

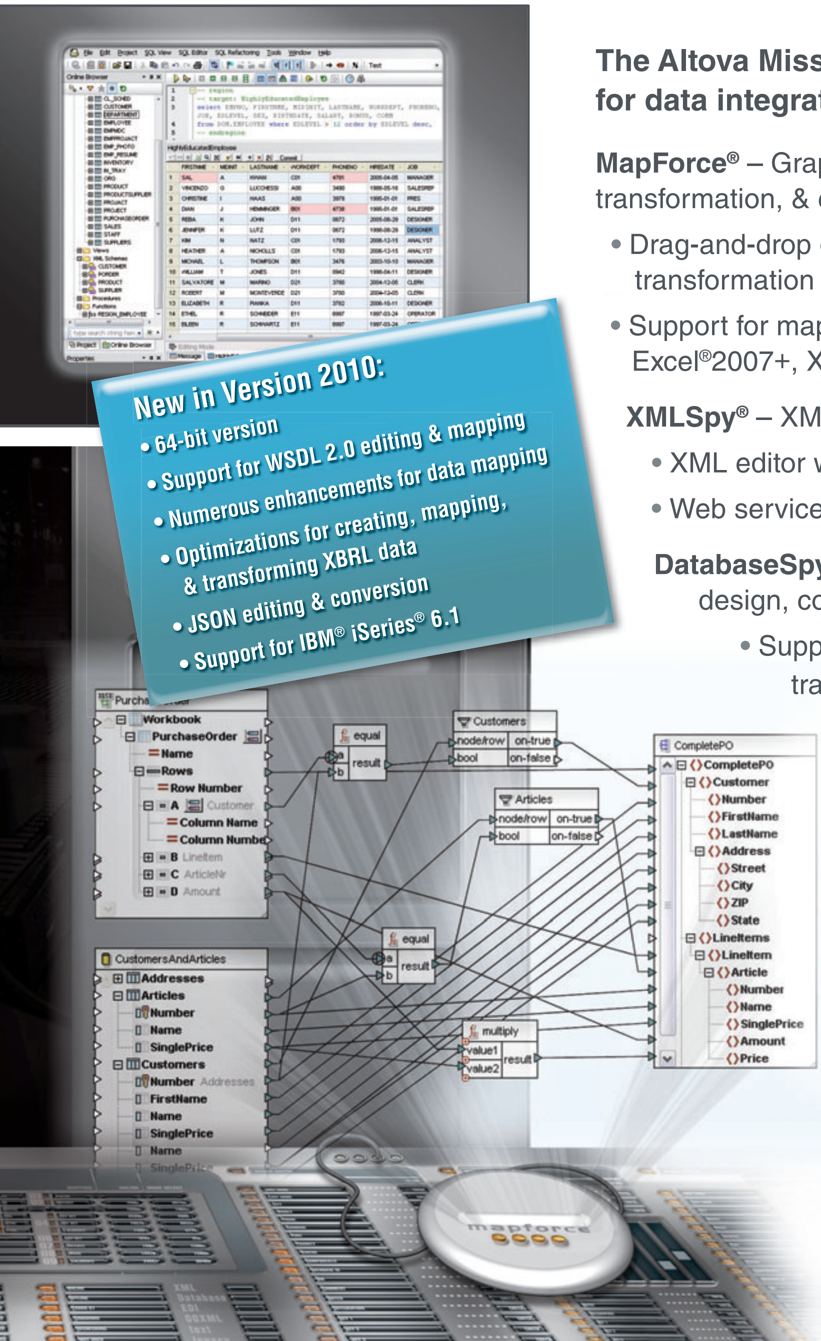
Figure 2 **Mapping Table Columns Directly to Entity Properties**



ALTOVA® missionkit®

Connect legacy technologies affordably with the complete set of data integration tools from Altova®

Experience how the Altova MissionKit®, the integrated suite of XML, data mapping, and database tools, can help you leverage existing technology and business software investments while integrating modern technologies – without breaking your budget.



New in Version 2010:

- 64-bit version
- Support for WSDL 2.0 editing & mapping
- Numerous enhancements for data mapping
- Optimizations for creating, mapping, & transforming XBRL data
- JSON editing & conversion
- Support for IBM® iSeries® 6.1

The Altova MissionKit includes multiple intelligent tools for data integration:

MapForce® – Graphical data mapping, transformation, & conversion tool

- Drag-and-drop data conversion with instant transformation & code generation
- Support for mapping XML, DBs, EDI, XBRL, Excel®2007+, XBRL, text files, Web services

XMLSpy® – XML editor and Web services tool

- XML editor with strong database integration
- Web services tool, JSON <> XML converter

DatabaseSpy® – multi-database query, design, comparison tool

- Support for all major relational databases and translation between DB types
- SQL editor, graphical database design & content editor

Download a 30 day free trial!

Try before you buy with a free, fully functional, trial from **www.altova.com**

Figure 3 **The SSDL Description of the Database Customer Table**

```
<EntityType Name="Customer">
  <Key>
    <PropertyRef Name="CustomerID" />
  </Key>
  <Property Name="CustomerID" Type="int" Nullable="false"
            StoreGeneratedPattern="Identity" />
  <Property Name="Title" Type="nvarchar" MaxLength="8" />
  <Property Name="FirstName" Type="nvarchar" Nullable="false"
            MaxLength="50" />
  <Property Name="MiddleName" Type="nvarchar" MaxLength="50" />
  <Property Name="LastName" Type="nvarchar" Nullable="false"
            MaxLength="50" />
  <Property Name="Suffix" Type="nvarchar" MaxLength="10" />
  <Property Name="CompanyName" Type="nvarchar" MaxLength="128" />
  <Property Name="SalesPerson" Type="nvarchar" MaxLength="256" />
  <Property Name="EmailAddress" Type="nvarchar" MaxLength="50" />
  <Property Name="Phone" Type="nvarchar" MaxLength="25" />
  <Property Name="ModifiedDate" Type="datetime" Nullable="false" />
  <Property Name="TimeStamp" Type="timestamp" Nullable="false"
            StoreGeneratedPattern="Computed" />
</EntityType>
```

namespace of the conceptual model's entity container. You're required to use the strongly typed names of both the Conceptual Schema Definition Language (CSDL) and SSDL types that are referenced in the expression.

As long as the results of the projection (an integer, a string and a string) match the schema of the target entity, the mapping will succeed. I've tried to use functions and concatenation within the projection—for example, (c.CustomerID, c.FirstName + c.LastName)—but this fails with an error stating that FUNCTIONs are not allowed. So I'm forced to use the FirstName and LastName properties and let the client deal with concatenation.

Placing the QueryView into the Metadata

You must place the QueryView expression within the EntitySetMapping element for the entity that goes inside the EntityContainerMapping in the metadata. **Figure 4** shows this QueryView (highlighted in yellow) in the raw XML of my EDMX file.

Now my CustomerNameAndID is part of my model and will be available to any consumer. And there is another advantage to the QueryView. Even though the goal of this QueryView is to create a read-only reference list, you can also update entities that are mapped using QueryViews. The context will track changes to CustomerNameAndID objects. Although Entity Framework is not able to auto-generate insert, update and delete commands for this entity, you can map stored procedures to it.

Reaping the Benefits of the QueryView

Now that you have the QueryView in the model, you don't need to depend on projections or anonymous types to retrieve these views of your data. In WCF Data Services, CustomerNameAndIDs becomes a valid entity set to query against, as shown here:

```
List<CustomerNameAndID> custPickList =
  context.CustomerNameAndIDs.ToList();
```

No messy projections. Better yet, you can create service operations in your custom WCF Services that are now able to return this strongly typed object without having to define new types in your application and project into them.

```
public List<CustomerNameAndID> GetCustomerPickList()
{
  using (var context = new AWEntities())
  {
    return context.CustomerNameAndIDs.OrderBy(
      c => c.LastName).ToList();
  }
}
```

Because of the limitation that prevents us from concatenating the first and last names in the QueryView, it's up to the developers who consume the service to do this concatenation on their end.

WCF RIA Services can also benefit from the QueryView. You may want to expose a method for retrieving a restaurant pick list from your domain service. Rather than having to create an extra class in the domain service to represent the projected properties, this RestaurantPickList entity is backed by a QueryView in the model, which makes it easy to provide this data:

```
public IQueryable<RestaurantPickList> GetRestaurantPickList()
{
  return context.RestaurantPickLists;
}
```

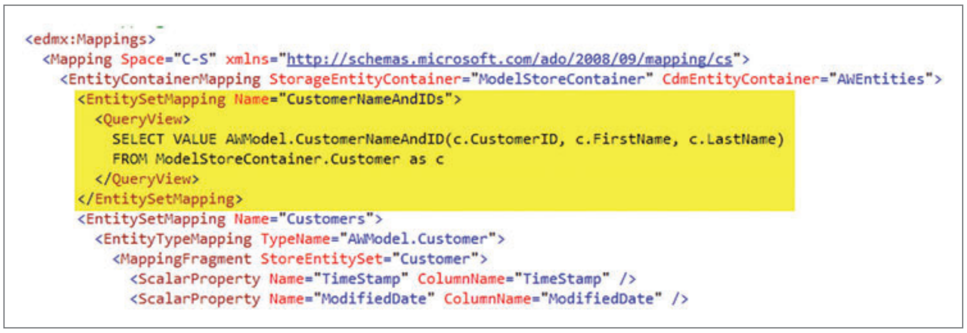
QueryViews or Projections—We've Got You Covered

Having the ability to project views over your data types is a huge benefit in querying, and it's a great addition to WCF Data Services. Even so, there are times when having access to these views, without having to project and without having to worry about sharing the result, will simplify some of your coding tasks.

One last note: With the introduction of foreign keys in the .NET Framework 4 version of Entity Framework, QueryView pick lists make even more sense because you can return read-only entities and simply use their properties to update foreign key properties in the entities you're editing. ■

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). Follow her on Twitter: julielerman.*

THANKS to the following technical expert for reviewing this article: *Alex James*



```
<edmx:Mappings>
  <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2008/09/mapping/cs">
    <EntityContainerMapping StorageEntityContainer="ModelStoreContainer" CdmEntityContainer="AWEntities">
      <EntitySetMapping Name="CustomerNameAndIDs">
        <QueryView>
          SELECT VALUE AWModel.CustomerNameAndID(c.CustomerID, c.FirstName, c.LastName)
          FROM ModelStoreContainer.Customer as c
        </QueryView>
      </EntitySetMapping>
      <EntitySetMapping Name="Customers">
        <EntityTypeMapping TypeName="AWModel.Customer">
          <MappingFragment StoreEntitySet="Customer">
            <ScalarProperty Name="TimeStamp" ColumnName="TimeStamp" />
            <ScalarProperty Name="ModifiedDate" ColumnName="ModifiedDate" />
```

Figure 4 **A QueryView in the Mappings Section**

Lightweight.
Intuitive.
Fast.

The solution that helps you to focus on your real job: developing great software.

Seamless task tracking and source control

Browse your tasks, check on progress, automatically retrieve the correct code base and know who's working on the same project files.

Full Visual Studio integration

Whether you're working with the GUI or the full Visual Studio integration, all PureCM features are available at your fingertips.

High performance, scalable

Private developer workspaces with fast checkin operations and full support for distributed development, even when offline.

VS 2010 ready

Agile planning and source control.
Get your free evaluation resources at
www.purecm.com/start10-1

PureCM

Alien approved for 20+ Light Years!
THE CONTROLS HAVE LANDED.
s2
s3
s4
s5
s6
s1
s7
June
July
April
May
June
November
December
ComponentOne
Studio Enterprise 2010
Get yours today at: COMPONENTONE.COM/HERE

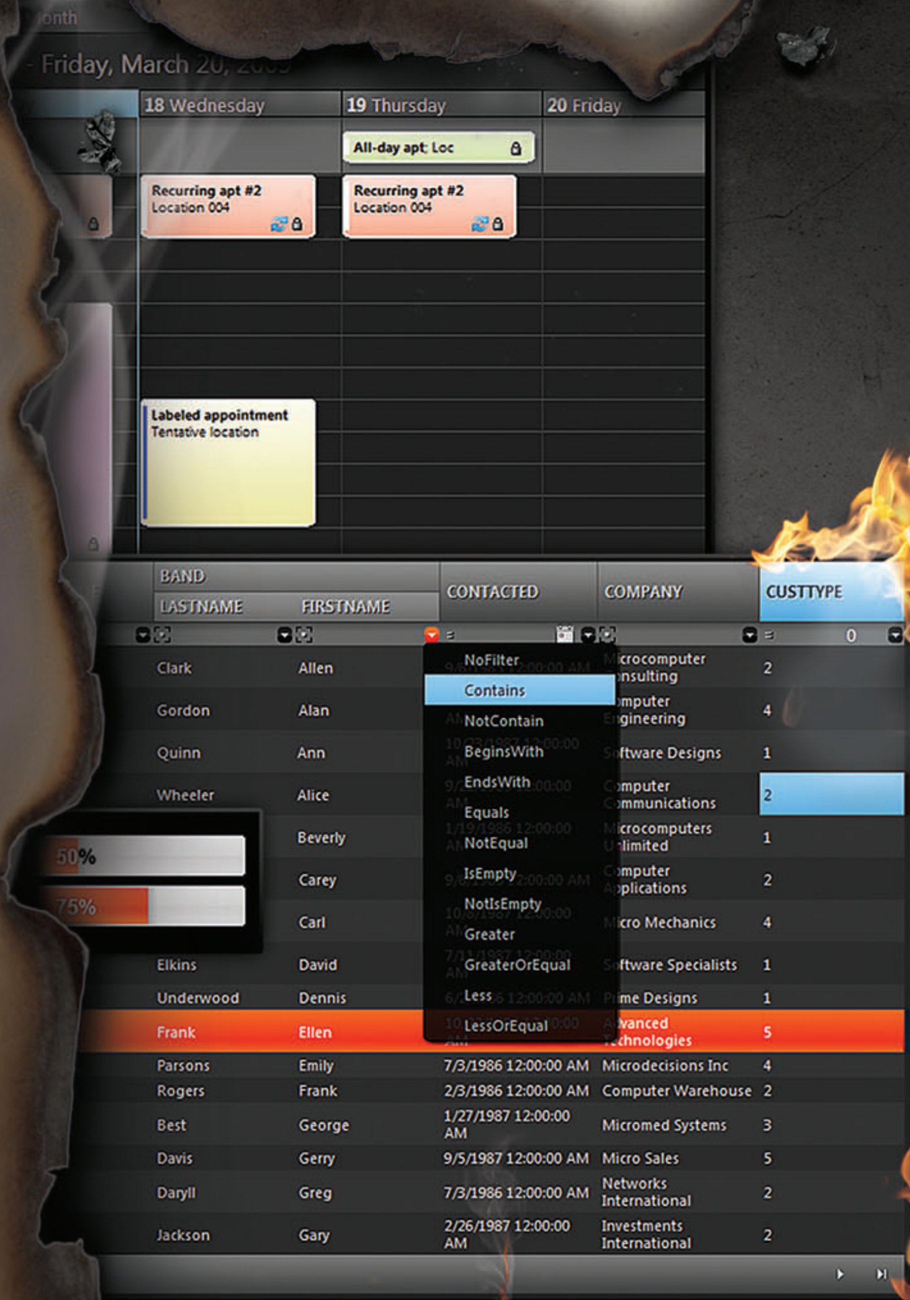
Sheela Castle	20	Peter Campbell 8287	Netherlands	Vanuatu
Irina Vera	37	Kenedy Avenue 9758	Botswana	Vatican City
Noela Meredith	39	5th Avenue 3139	India	Macau --
Michael Days	48	10th Avenue 7141	Namibia	Australia
Alvaro Johansen	21	13th Street 9183	Australia	Cuba
Noela Vera	37	Missisipi 8428	Cape Verde	Gabon
Patric Johnson	19	Peter Campbell 9747	Japan	Tunisia

100'S OF CONTROLS, 7 PLATFORMS

Featuring WinForms • WPF • ASP.NET AJAX • Silverlight • iPhone • Mobile • ActiveX

Grids • Charting • Reporting • Scheduling • Menus and Toolbars • Ribbon • Data Input • Editors • PDF

VISUAL STUDIO 2010-READY



BAND		CONTACTED	COMPANY	CUSTTYPE
LASTNAME	FIRSTNAME			
Clark	Allen	[illegible]	Microcomputer nsulting	2
Gordon	Alan	[illegible]	mputer ngineering	4
Quinn	Ann	[illegible]	ftware Designs	1
Wheeler	Alice	[illegible]	omputer ommunications	2
[illegible]	Beverly	[illegible]	icrocomputers nlimited	1
[illegible]	Carey	[illegible]	omputer pplications	2
[illegible]	Carl	[illegible]	icro Mechanics	4
Elkins	David	[illegible]	ftware Specialists	1
Underwood	Dennis	[illegible]	me Designs	1
Frank	Ellen	[illegible]	vanced chnologies	5
Parsons	Emily	7/3/1986 12:00:00 AM	Microdecisions Inc	4
Rogers	Frank	2/3/1986 12:00:00 AM	Computer Warehouse	2
Best	George	1/27/1987 12:00:00 AM	Micromed Systems	3
Davis	Gerry	9/5/1987 12:00:00 AM	Micro Sales	5
Daryll	Greg	7/3/1986 12:00:00 AM	Networks International	2
Jackson	Gary	2/26/1987 12:00:00 AM	Investments International	2

Experience the rich, new features of **Visual Studio 2010** along with the ability to:

- Improve your Web Form apps in minutes with AJAX 4.0 controls.
- Style your Silverlight applications using visual brush creation and ComponentOne ClearStyle technology.
- Add data visualization such as charts and gauges in WinForms and ASP.NET AJAX.
- Add docking and floating capabilities to WPF windows.

ASP.NET · AJAX · JQUERY · CSS · XHTML · JSON · W3C STANDARDS · CLIENT SIDE OBJECT MODEL (CSOM) · ANIMATION · THEME DESIGNER (FOXY)

Studio for ASP.NET AJAX 2010

FUELED BY THE NEW **A2 FRAMEWORK**

Sold separately. Also part of Studio Enterprise.

ComponentOne® COMPONENTONE.COM/AJAX

ComponentOne Sales: 1.800.858.2739 or 1.412.681.4343

Building Custom Players with the Silverlight Media Framework

Ben Rush

Streaming media has become ubiquitous on the Web. It seems like everyone—from news sites to social networks to your next-door neighbor—is involved in the online video experience. Due to this surge in popularity, most sites want to present high-quality video—and often high-quality *bandwidth-aware* video—to their consumers in a reliable and user-friendly manner.

A key element in the online media delivery experience is the player itself. The player is what the customer interacts with, and it drives every element of the user's online experience. With so much attention centered on the player, it's no surprise that modern, Web-based media players have become a great deal more complicated to implement than they were even a couple years ago. As a result, developers need a robust framework on which they can build their players.

This article is based on the Smooth Streaming Player Development Kit beta 2. All information is subject to change.

This article discusses:

- Getting started with SMF
- Creating and displaying the player
- Logging, settings and events
- Showing recommended video buttons

Technologies discussed:

Silverlight Media Framework, Smooth Streaming Player Development Kit

Code download available at:

code.msdn.microsoft.com/mag201005CustPlay

The Silverlight Media Framework (SMF) is an open source project that was released by Microsoft at the 2009 Microsoft Professional Developers Conference. It is an extensible and highly scalable Silverlight video framework that directly answers the need for a stable core upon which developers and designers can create their own players. The code at the center of the Silverlight Media Framework has been refined based on lessons learned from the NBC Olympics and Sunday Night Football Web video projects.

This article will explain the basic elements of SMF, demonstrate how you can integrate SMF into your own player projects and walk you through a simple project that uses SMF to create a custom player experience. I'll show you how to use the logging, settings, and event-handling features of SMF. Finally, I'll create a player application that displays suggested videos for further viewing when the current video ends.

Getting Started with SMF

To get started, the first thing you'll want to do is download the framework from Codeplex (smf.codeplex.com). You also need to download the Smooth Streaming Player Development Kit (iis.net/expand/smoothplayer) and reference it in any projects using SMF. The Smooth Streaming Player Development Kit is not part of SMF—it's a completely separate, closed-source component. However, SMF leverages a core set of functionality from the kit, in particular the video player itself. As of the writing of this article, the Smooth Streaming Player Development Kit is in beta 2.

SMF consists of a number of Microsoft .NET assemblies (as shown in **Figure 1**), each a different functional part of the overall framework.

The core assembly is Microsoft.SilverlightMediaFramework.dll, which comprises a number of utility classes and types referenced throughout the rest of the framework. When using any aspect of SMF, you must also reference the Microsoft.SilverlightMediaFramework.dll assembly.

The Microsoft.SilverlightMediaFramework.Data namespace provides helper classes for consuming data external to the player and for encapsulating data within the player. The data can be general, with any form, but it can also be settings information for the player itself. There's another namespace, Microsoft.SilverlightMediaFramework.Data.Settings, for types representing and dealing with player settings.

Apart from data used for settings, the type within the Data namespace you'll most likely interact with is the out-of-stream *DataClient* class, which can retrieve data from an external source. You reference this assembly if you want to download and use data external to the player.

The SMF player includes the robust Microsoft.SilverlightMediaFramework.Logging framework that uses a callback-style paradigm in which writing to the logging infrastructure raises events. You register your own callback methods with the logging system, and these callbacks carry out additional operations once invoked—such as posting information to a Web service or displaying information to a text box. You reference this assembly if you wish to use the built-in logging facilities of SMF.

The Microsoft.SilverlightMediaFramework.Player assembly implements the player itself. It also provides a number of controls the player relies on, such as a scrubber, volume control and timeline markers. The default SMF player is sleek and clean, a great starting point for any project requiring a Silverlight player. However, central to all controls defined within SMF is the notion of control templating, so each control can be themed by using tools such as Expression Blend or Visual Studio.

Building and Referencing SMF

SMF downloads as a single .zip file in which you'll find a solution file, a project for each output library, and test projects for running and verifying the player itself.

SMF relies on the Smooth Streaming Player Development Kit. To reference the kit, move the Smooth Streaming assembly (Microsoft.Web.Media.SmoothStreaming.dll) into the \Lib folder of the SMF project.

Next, open the SMF solution in Visual Studio and build it, creating all the assemblies needed to leverage the framework. To verify that everything executes as expected, press F5 to begin debugging. The solution will build and the Microsoft.SilverlightMediaFramework.Test.Web target will execute, presenting you with the default SMF player streaming a "Big Buck Bunny" video (see **Figure 2**). Note how complete the default player already is, with a position element for scrubbing, play/stop/pause buttons, volume controls, full screen controls and so forth.

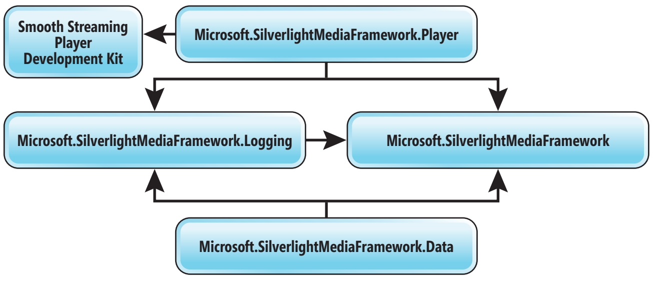

Smooth Streaming Player Development Kit
Microsoft.SilverlightMediaFramework.Player
Microsoft.SilverlightMediaFramework.Logging
Microsoft.SilverlightMediaFramework
Microsoft.SilverlightMediaFramework.Data

Figure 1 **The Silverlight Media Framework Assemblies**

The next step is to create your own separate Silverlight project and leverage SMF from within it. In Visual Studio click File | New | Project | Silverlight Application. Call the solution SMFPlayerTest and click OK. A modal dialog will pop up, asking whether you wish to host the Silverlight application in a new Web site. Click OK and you'll see a basic Silverlight application solution consisting of two projects, SMFPlayerTest and SMFPlayerTest.Web.

The final step is to reference the Smooth Streaming Player Development Kit and SMF assemblies from your newly created project. Copy the output SMF assemblies and Smooth Streaming Player Development Kit from the SMF solution's Debug folder and paste them into your new project as shown in **Figure 3**. Your new solution now includes all the assembly references required to take full advantage of the SMF.

Displaying the Player

To begin using the SMF, include the SMF player's namespace within your MainPage.xaml page. This ensures that all references resolve properly:

```
xmlns:p="clr-namespace:Microsoft.SilverlightMediaFramework.
Player;assembly=Microsoft.SilverlightMediaFramework.Player"
```

Now insert player's XAML within the page's LayoutRoot Grid control.

```
<Grid x:Name="LayoutRoot">
  <p:Player>
  </p:Player>
</Grid>
```

Pressing F5 will launch the project and bring up the SMF player. However, because the player hasn't been told what to play, it does nothing. All you get is a player with no content to play.

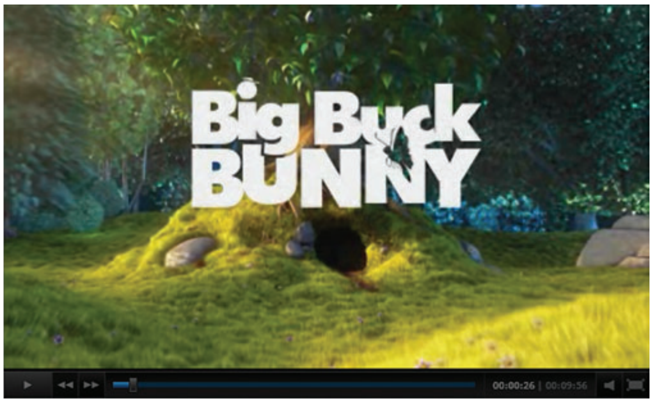


Figure 2 **The SMF Player and the Big Buck Bunny Video**

SMF uses SmoothStreamingMediaElement (from the Smooth Streaming Player Development Kit) to play video. From SmoothStreamingMediaElement, SMF inherits its own player, called CoreSmoothStreamingMediaElement. This object is required if you want the player to stream content. Be sure to set the SmoothStreamingSource property to a valid smooth streaming media URL:

```
<Grid x:Name="LayoutRoot">
  <p:Player>
    <p:CoreSmoothStreamingMediaElement
      AutoPlay="True"
      SmoothStreamingSource="replace with address to content here"/>
  </p:Player>
</Grid>
```

As mentioned earlier, Microsoft provides the "Big Buck Bunny" sample video stream, which developers can use to test Silverlight projects. To use this test stream, set the SmoothStreamingSource property on the CoreSmoothStreamingMediaElement to:

```
http://video3.smoothhd.com.edgesuite.net/ondemand/Big%20Buck%20Bunny%20
Adaptive.ism/Manifest
```

Once again, press F5 to build and run the project. The browser will execute with the same player as before, but this time the "Big Buck Bunny" video will begin streaming moments after the player has fully loaded. If your task was to create a basic Silverlight player to stream content, you've done it.

However, the SMF offers quite a bit more than we've seen thus far. Let's add some basic logging.

Logging in the Player

Logging in SMF is simple—whenever an event is logged, it raises a LogReceived event. You register an event handler for this event, and thereby receive a notification for each logging event as it's raised. What you do with the notification is up to you; you can display it in a new window within the player, filter the events and notify a Web service whenever a certain event gets raised, or do whatever is necessary for your scenario.

The LogReceived event is statically defined on the Logger class itself (defined within Microsoft.SilverlightMediaFramework.Logging.dll), so it's possible to register for logging events anywhere within the project. Here's an example of registering for and defining the event handler within the MainPage.xaml file of the SMFPlayerTest project:

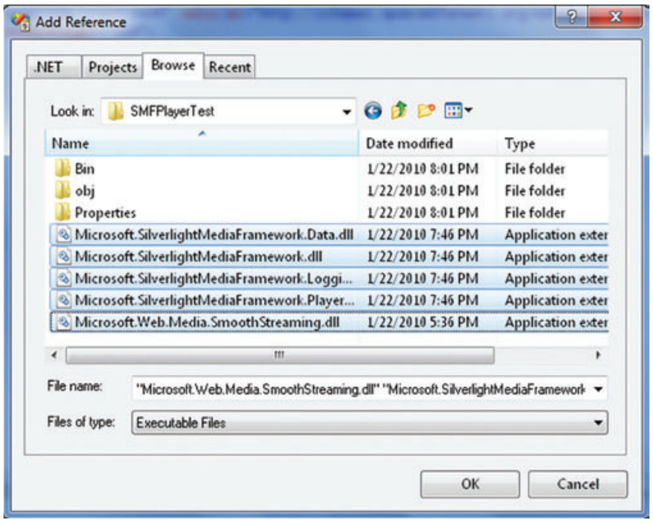
```
public partial class MainPage : UserControl {
  public MainPage() {
    InitializeComponent();

    Logger.LogReceived +=
      new EventHandler<SimpleEventArgs<Log>>(
      Logger_LogReceived);
  }

  void Logger_LogReceived(object sender,
    Microsoft.SilverlightMediaFramework.SimpleEventArgs<Log> e) {
    throw new NotImplementedException();
  }
}
```

SMF raises quite a few events out of the box. To see them, create a breakpoint within the Logger_LogReceived method and run the player once again in Debug mode. Almost immediately your breakpoint will get hit, allowing you to step through the method's parameters and see the information passed to it.

Log event data is packaged within a special messaging object whose type must inherit from an abstract class named Log. This abstract Log type has three properties: Sender, Message and TimeStamp. Sender references the object that raised the event. Message is an


Add Reference

.NET | Projects | Browse | Recent

Look in: SMFPlayerTest

Name	Date modified	Type
Bin	1/22/2010 8:01 PM	File folder
obj	1/22/2010 8:01 PM	File folder
Properties	1/22/2010 8:01 PM	File folder
Microsoft.SilverlightMediaFramework.Data.dll	1/22/2010 7:46 PM	Application exter
Microsoft.SilverlightMediaFramework.dll	1/22/2010 7:46 PM	Application exter
Microsoft.SilverlightMediaFramework.Loggi...	1/22/2010 7:46 PM	Application exter
Microsoft.SilverlightMediaFramework.Player...	1/22/2010 7:46 PM	Application exter
Microsoft.Web.Media.SmoothStreaming.dll	1/22/2010 5:36 PM	Application exter

File name: "Microsoft.Web.Media.SmoothStreaming.dll" "Microsoft.SilverlightMediaFramework

Files of type: Executable Files

OK | Cancel

Figure 3 **Referencing the Required Assemblies**

object of type System.String that holds the text for the logging event. TimeStamp simply holds the date and time at which the logging object was first instantiated. The SimpleEventArgs<> object passed as the second parameter to your event handler holds a reference to the Log object through its Result property.

To raise a log event, all that's required is to instantiate a type that inherits from the Log base class, then pass this type to the statically defined Log method on the Logger type. The framework supplies a DebugLog class that already inherits from the Log base type. What's special about the DebugLog type, however, is that if the libraries being referenced by your Silverlight project were created under a Debug build of the SMF, passing a DebugLog type to the SMF logging framework will raise a corresponding logging event (and therefore invoke your event handlers). On the other hand, a Release build of the SMF will ignore any call to the Log method that gets passed the DebugLog class. In short, if you have debugging statements you only want to use Debug builds, with the DebugLog object as the log event argument; otherwise you will need to construct your own type that inherits from the abstract Log type.

Here's an example that raises a Listening event through the SMF event system by instantiating a DebugLog object and passing it to the Logger's static Log method (be sure your Smooth Streaming Player Development Kit files were built under Debug settings):

```
public MainPage() {
  InitializeComponent();

  Logger.LogReceived +=
  new EventHandler<SimpleEventArgs<Log>>(
    Logger_LogReceived);

  Logger.Log(new DebugLog {
    Message = "Listening!", Sender = this });
}
```

Inheriting from the Player Class

Although logging is a central feature of the player, the SMF playback features are only accessible when you inherit from and begin extending the SMF Player type itself.

To see how this works, you need to create a new class called SMFPlayer that inherits from the Player type.

/n software

We didn't invent the Internet...

...but our components help *you* power the apps that bring it to business.

applications

powered by

connectivity

powered by

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

To learn more please visit our website → www.nsoftware.com

Microsoft .NET, | Java | ActiveX | C++ | Delphi | C++ Builder | PHP | Objective-C | Windows | Windows Mobile | Mac OS X | Linux/Unix | iPhone

The new SMFPlayer class looks like this:

```
namespace SMFPlayerTest {
  public class SMFPlayer : Player {
    public override void OnApplyTemplate() {
      base.OnApplyTemplate();
    }
  }
}
```

Every FrameworkElement type (such as Player in SMF) has an OnApplyTemplate method that is called whenever the ApplyTemplate event is raised. This method often serves as a useful starting point when initializing a FrameworkElement type.

In this case, I override the default OnApplyTemplate method from within the new SMFPlayer class. To demonstrate that the new SMFPlayer type is executed instead of the default Player type, you can set a breakpoint within the override. When you debug the player in Visual Studio, this breakpoint will be enountered when Silverlight executes the SMFPlayer.

Built into the SMF player is a robust logging framework.

Now update the MainPage.xaml file to use the new player class. First, include the player's namespace in the list of namespaces already referenced (just as you did the player namespace earlier):

```
xmlns:smf="clr-namespace:SMFPlayerTest"
```

Then simply update the Player tags within the XAML to use SMFPlayer instead of Player:

```
<Grid x:Name="LayoutRoot">
  <smf:SMFPlayer>
    <p:CoreSmoothStreamingMediaElement
      AutoPlay="true"
      SmoothStreamingSource="http://..."/>
  </smf:SMFPlayer>
</Grid>
```

Next, instantiate a DebugLog class and pass it to the Log method as shown earlier. Doing so will fire the event for which you previously registered an event handler:

```
public override void OnApplyTemplate() {
  Logger.Log(new DebugLog {
    Message = "Hello from OnApplyTemplate!",
    Sender = this
    });

  base.OnApplyTemplate();
}
```

To listen specifically for this event from within the event handler, filter the Message property of the DebugLog object itself. In this example, look for any message that contains "OnApplyTemplate":

```
void Logger_LogReceived(
  object sender, SimpleEventArgs<Log> e) {
  if (e.Result.Message.Contains("OnApplyTemplate")) {
    return;
  }
}
```

Using Settings Data

A mature framework for dealing with settings is crucial to most large-scale software projects. The code for handling settings in SMF is built on the Microsoft.SilverlightMediaFramework.Data.dll assembly, which allows you to download generic, external data. The settings layer of SMF uses this infrastructure to reach out and download a specially formatted XML settings file hosted on a Web server. Once the settings data has been successfully downloaded and read, the SMF settings layer encapsulates it with a SettingsBase object whose methods are then used to retrieve the settings values.

The SettingsBase class, as the name suggests, serves as a base for a more specific class that can provide strongly typed access to your settings values. Here's an example of a class that inherits from SettingsBase. It has two properties, one for retrieving a video player source URL and another for retrieving a Boolean value that indicates whether the video player should start automatically or wait for the viewer to press the play button:

```
namespace SMFPlayerTest {
  public class SMFPlayerTestSettings : SettingsBase {
    public Uri VideoPlayerSource {
      get { return new Uri(
        GetParameterValue("videoSource")); }
    }

    public bool? AutoStartVideo {
      get { return GetParameterBoolean(
        "autoStart"); }
    }
  }
}
```

The property methods use functions implemented by the SettingsBase class to inspect the underlying collection of settings name/value pairs loaded into the type (through a mechanism discussed shortly). This provides a type-safe and IntelliSense-friendly method of retrieving settings information.

Now create a new XML file in the SMFPlayerTest.Web project, name it SMFPlayerSettings.xml, and add the following to it:

```
<?xml version="1.0" encoding="utf-8" ?>
<settings>
  <Parameters>
    <Parameter
      Name="videoSource"
      Value="http://video3.smoothhd.com.edgesuite.net/ondemand/Big%20
Buck%20Bunny%20Adaptive.ism/Manifest"/>
    <Parameter Name="autoStart" Value="True"/>
  </Parameters>
</settings>
```

Next, create a SettingsClient object into which you'll load the settings XML. SettingsClient takes a URI pointing to the settings file:

```
m_settingsGetter = new SettingsClient(
  new Uri("http://localhost:10205/SMFPlayerSettings.xml"));
```

The process of retrieving the settings data is asynchronous, so a callback method must be assigned to the RequestCompleted method on SettingsClient:

```
m_settingsGetter.RequestCompleted +=
  new EventHandler<SimpleEventArgs<SettingsBase>>
  (m_settingsGetter_RequestCompleted);
```

The last step is to invoke the parameterless Fetch method on the SettingsClient object. When the data is retrieved, the settingsGetter_RequestCompleted event handler will be invoked and a SettingsBase object will be passed to it:

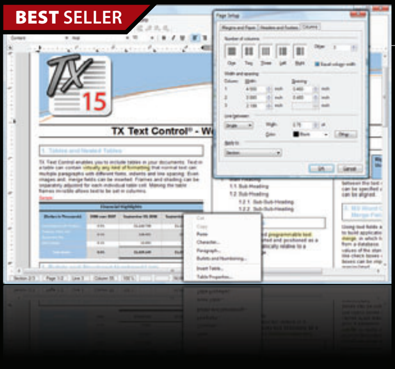
```
void m_settingsGetter_RequestCompleted(
  object sender, SimpleEventArgs<SettingsBase> e) {

  SettingsBase settingsBase = e.Result;
  return;
}
```

The SettingsBase object passed to the settingsGetter_RequestCompleted method is loaded with the name/value pairs parsed for you by the underlying framework from the file SMFPlayerSettings.xml.

ComponentSource®
The Definitive Source of Software Components
www.componentsource.com

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

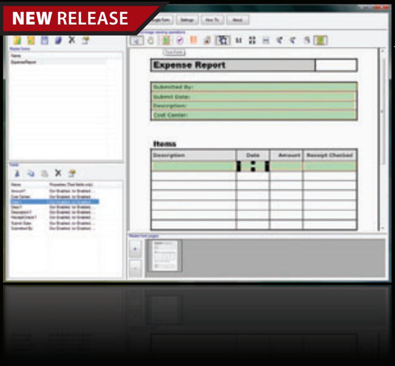
FusionCharts | from $195.02

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

NEW RELEASE



LEADTOOLS Recognition SDK | from $3,595.50

LEAD TECHNOLOGIES INCORPORATED

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
- Includes comprehensive confidence reports to assess performance

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

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

MasterCard AMERICAN EXPRESS VISA DISCOVER

GSA Schedule
Contract GS-35F-0188R

In order to load this data into your SMFPlayerTestSettings object, you simply call the Merge method, which merges settings information from one SettingsBase-derived object with that of another:

```
SettingsBase settingsBase = e.Result;
m_settings.Merge(settingsBase);

this.mediaElement.SmoothStreamingSource =
  m_settings.VideoPlayerSource;
this.mediaElement.AutoPlay =
  (bool)m_settings.AutoStartVideo;

return;
```

You no longer have to hard-code the AutoPlay and SmoothStreamingSource properties on the CoreSmoothStreamingMediaElement within the page XAML, because the player settings are being downloaded from within the OnApplyTemplate method. This is all you need for the player XAML:

```
<Grid x:Name="LayoutRoot">
  <smf:SMFPlayer>
    <p:CoreSmoothStreamingMediaElement/>
  </smf:SMFPlayer>
</Grid>
```

When you run the player, all the settings data will load, the callback will load the values into the player's media element, and the video will begin to stream just as it did before.

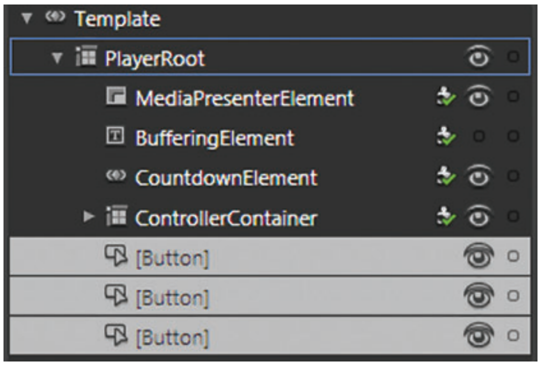


Figure 4 **Button Controls Added to the Control Tree**

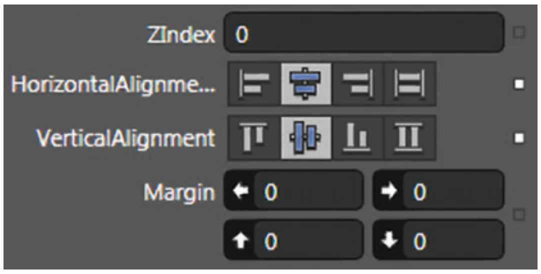


Figure 5 **Setting Button Control Alignment**

Extending the SMF Player

On many popular video sites, when video playback has completed, you see a list of similar or recommended videos. To illustrate how easy it is to extend the SMF player, let's walk through the steps to build a similar suggested-viewing feature into the SMFPlayerTest project.

Start by adding an x:Name attribute to the Player element in the MainPage.xaml file:

```
<Grid x:Name="LayoutRoot">
  <smf:SMFPlayer x:Name="myPlayer">
    <p:CoreSmoothStreamingMediaElement/>
  </smf:SMFPlayer>
</Grid>
```

This makes it easier to refer to the SMFPlayer object by name within both Visual Studio and Expression Blend.

Now, right-click on the MainPage.xaml file in Solution Explorer and select Open in Expression Blend. Expression Blend 3 will launch and display a design interface to the SMF player. In the Objects and Timeline section, you'll find a myPlayer node in the tree of visual objects that corresponds to the name given to the SMFPlayer object previously. The goal is to create a template for SMFPlayer, then to add three Suggestion buttons to the template. By using a template in Expression Blend, you can add, edit or remove controls built into the player itself.

To create a template, right-click myPlayer in the Objects and Timeline window and select Edit Template | Edit a Copy. A Create Style Resource dialog will be displayed, click OK. To insert the three buttons on top of the video player, double-click the button icon in the Tools window for each button you want to add. Three buttons should now be visible in the tree of controls that make up the player template (see **Figure 4**).

Select all three buttons in the tree, go to the properties window for the controls and set the horizontal and vertical alignment to be centered (see **Figure 5**), thus aligning the buttons down the center and middle of the video player.

The buttons are the default size and lie on top of each other. Set the width of each button to 400, and the height to 75. Next, adjust the margins so that one button has a 175-pixel offset from the bottom, another 175-pixel offset from the top and the last has no margin offsets at all. The end result will look like **Figure 6**.

To verify the buttons have been properly placed on the player, save all open files in Expression Blend and return to Visual Studio. Visual Studio may prompt you to reload documents that were changed by Expression Blend. If so, click OK. From within Visual Studio, press F5 to relaunch the SMF player in Debug mode. The player should now appear with three buttons aligned down the center of the video screen as shown in **Figure 7**.

Hooking up Event Handlers

Event handlers must now be associated with the buttons. To reference the buttons from code, you need to assign names to them, which you do via the Name text box in the Properties tab. For simplicity, name the buttons Button1, Button2 and Button3. When you're done, the Objects and Timeline window should update and display the button names adjacent to the button icons in the visual tree.

Within the Properties tab for each button you'll find an Events button that's used to assign event handlers for a visual component. Select one of the buttons, click the Event button within the Properties tab, and double-click the Click text box to auto-generate an event handler within the MainPage.xaml.cs. The properties window for each button will now have an event handler assigned to its Click event (see **Figure 8**), and the MainPage.xaml.cs file will have event handlers assigned to each button's Click event.

You can now debug the player. Clicking any of the buttons on the screen will raise a Click event, which is now handled by the auto-generated methods within MainPage.xaml.cs.

Suggested Videos

Now let's use these buttons to enable the suggested video feature. The following XML will represent the suggestions:

```
<?xml version="1.0" encoding="utf-8" ?>
<Suggestions>
  <Suggestion DisplayName="A suggestion" Url=""/>
  <Suggestion DisplayName="Another suggestion" Url=""/>
  <Suggestion DisplayName="My final suggestion" Url=""/>
</Suggestions>
```

I KNOW A PLACE THAT'S DIFFERENT, BUT FAMILIAR

Code in the cloud … with the development technologies you already know

With **Windows Azure™**, you have a flexible, on-demand infrastructure that taps into all the IT resources you need. The advantages?

You have the freedom to focus.
With the cloud as your development platform, you can focus on what you do best: create applications, without the restrictions on performance, capacity, or deployment that come with using your own hardware rather than the cloud.

You can code in the language you choose.
Windows Azure enables you to build applications in multiple languages, including .NET, PHP, and Java.

You're already equipped for the cloud.
You can start developing applications for Windows Azure right away, using familiar tools such as Microsoft® Visual Studio® 2008.

Find out more.
See how Windows Azure can help get you coding in the cloud. Plus, check out some offers exclusive to MSDN® subscribers. Go to **www.windowsazure.com/msdn** now.

Windows Azure™

Microsoft®

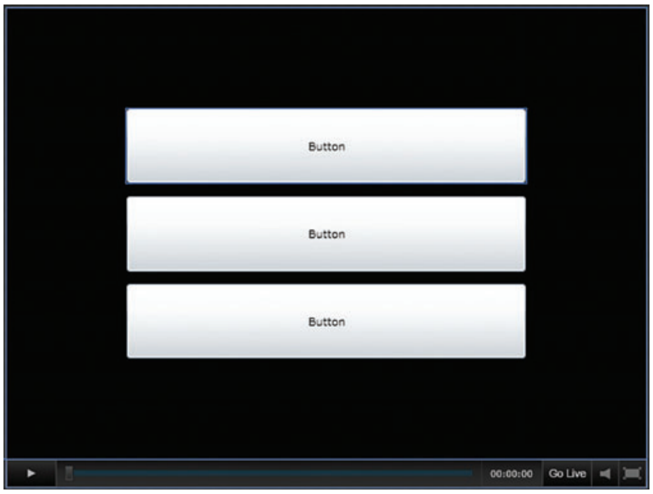


Figure 6 **The Centered Buttons in Expression Blend**

The value of the Url attribute will specify the video the player is to load when the button is clicked, and the DisplayName attribute is the text to be written on the button. Save this file with the name Suggestions.xml in the SMFPlayerTest.Web project.

The DataClient type (within the Microsoft.Silverlight-MediaFramework.Data namespace) will be used to download the XML document and to represent the content in a type-safe manner. To represent each Suggestion read from the XML file in a strongly typed fashion, create a class called SMFPlayerTestSuggestion in your Silverlight project:

```
namespace SMFPlayerTest {
  public class SMFPlayerTestSuggestion {
    public string DisplayName;
    public Uri Url;
  }
}
```

DataClient, like SettingsBase, is intended to be derived from by a class that enables a strongly typed representation of the data from the XML content (in this case, an array of SMFPlayerTest-Suggestion objects).

Create another class file within the SMFPlayerTest project called SMFPlayerTestDataClient:

```
namespace SMFPlayerTest {
  public class SMFPlayerTestDataClient :
    DataClient<SMFPlayerTestSuggestion[]> {

    public SMFPlayerTestDataClient(Uri Url) : base(Url) { }

    protected override void OnRequestCompleted(
      object sender, SimpleEventArgs<string> e) {

      throw new NotImplementedException();
    }
  }
}
```

SMFPlayerTestDataClient inherits from DataClient and sets its template argument to an array of SMFPlayerTestSuggestion types. The DataClient base class provides all the necessary asynchronous networking logic to go online and download the external XML file. Once the content has been downloaded, however, the DataClient base will invoke OnRequestCompleted and expect all processing of the XML data to take place then. In other words, the DataClient base class downloads the content, but the implementer is responsible for doing something with it.

Here's a more complete implementation of OnRequestCompleted:

```
protected override void OnRequestCompleted(
  object sender, SimpleEventArgs<string> e) {

  XDocument doc = XDocument.Parse(e.Result);
  List<SMFPlayerTestSuggestion> suggestions =
    new List<SMFPlayerTestSuggestion>();
  foreach (XElement element in doc.Descendants("Suggestion")) {
    suggestions.Add(new SMFPlayerTestSuggestion {
      DisplayName = element.Attribute("DisplayName").GetValue(),
      Url = element.Attribute("Url").GetValueAsUri()
    });
  }

  base.OnFetchCompleted(suggestions.ToArray());
}
```

For the sake of simplicity, I've used LINQ to XML in this implementation to parse the required elements and attributes in the XML. Once the DisplayName and Url attribute values from each Suggestion node have been retrieved, a SMFPlayerTestSuggestion object is instantiated and the values are assigned.

The final step is the invocation of OnFetchCompleted event. Outside consumers of SMFPlayerTestDataClient may register event handlers to the FetchCompleted event to be notified when the suggested video data has been downloaded. Because OnRequestCompleted has packaged the XML data in a type-safe manner, each event handler will receive a handy array of SMFPlayerTestSuggestion objects, one for each Suggestion element in the XML document the DataClient base class downloaded.

The underlying DataClient provides a method called Fetch that, once invoked, begins the process of asynchronously downloading content. To begin downloading the suggestion data when the video has ended, attach an event handler called mediaElement_MediaEnded to the MediaEnded event on the MediaElement object:

```
void mediaElement_MediaEnded(
  object sender, RoutedEventArgs e) {

  m_client = new SMFPlayerTestDataClient(
    new Uri("http://localhost:10205/Suggestions.xml"));
  m_client.FetchCompleted +=
    new EventHandler<SimpleEventArgs<
    SMFPlayerTestSuggestion[]>>(m_client_FetchCompleted);
  m_client.Fetch();
}
```

The mediaElement_MediaEnded method creates an instance of the SMFPlayerTestDataClient type, assigns another event handler to the FetchCompleted event, and then invokes Fetch to begin the

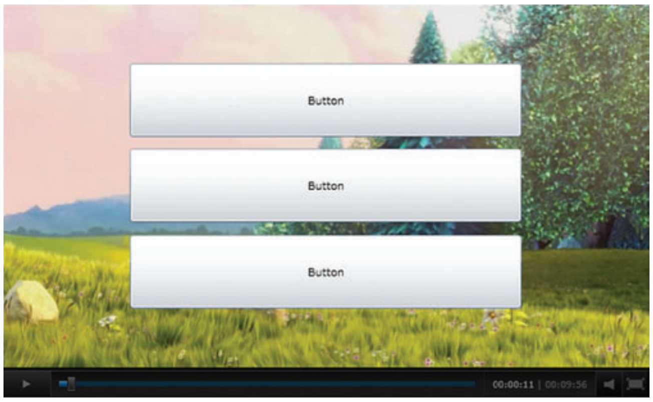


Figure 7 **The Centered Buttons in the SMF Player**

download process. The FetchCompleted handler will be invoked by the call to OnFetchCompleted implemented previously within OnRequestCompleted (which is invoked by the DataClient base type once the content has downloaded).

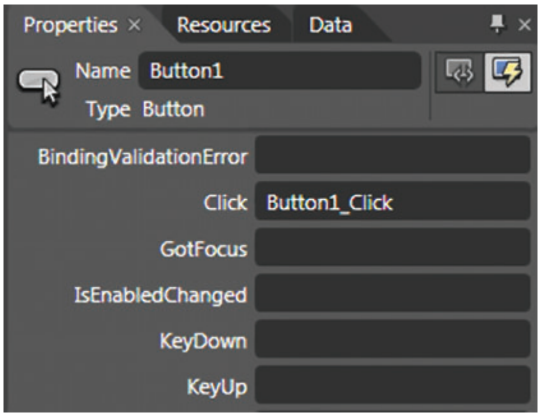
The implementation of suggestion_FetchCompleted, registered within mediaElement_MediaEnded, takes the strongly typed array of Suggestion data and assigns one Suggestion to each button:

```
void m_client_FetchCompleted(
  object sender, SimpleEventArgs<
  SMFPlayerTestSuggestion[]> e) {
  for (int c = 1; c <= 3; c++) {
    Button btn = (Button)GetTemplateChild(
      "Button" + c.ToString());
    btn.Tag = e.Result[c - 1].Url;
    btn.Content =
      e.Result[c - 1].DisplayName;
  }
}
```

GetTemplateChild, a method on the underlying FrameworkElement type, gets a reference to each of the buttons defined in the MainPage XAML. For each button, the display text is assigned to the Content property, and the URI is assigned to the Tag property. Each button's click event handler can then pull the URI from the Tag property and assign the URL to the player's MediaElement to play the stream:

```
private void Button1_Click(
  object sender, System.Windows.RoutedEventArgs e) {

  Uri redirectUrl = (Uri)((Button)sender).Tag;
  myPlayer.MediaElement.SmoothStreamingSource =
    redirectUrl;
}
```



Name Button1
Type Button
BindingValidationError
Click Button1_Click
GotFocus
IsEnabledChanged
KeyDown
KeyUp

Figure 8 **Setting the Event Handler**

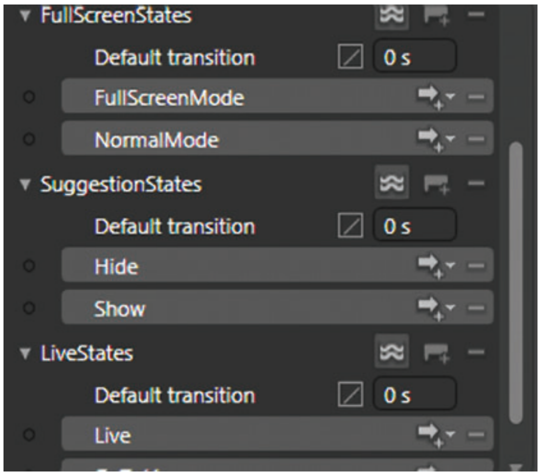


Figure 9 **Visual States for SuggestionStates**

Showing the Buttons

The final step is to hide the buttons until the currently streaming video has ended, at which point the buttons become visible. Once a user clicks a button, the buttons are hidden again.

Within Visual Studio, edit the SMFPlayer class by decorating it with two TemplateVisualState attributes:

```
[TemplateVisualState(Name = "Hide", GroupName = "SuggestionStates")]
[TemplateVisualState(Name = "Show", GroupName = "SuggestionStates")]
public class SMFPlayer : Player
```

TemplateVisualState is a fascinatingly powerful attribute that defines visual states under which an object may exist. Once a visual state becomes active, Silverlight will update properties of visual elements belonging to the class as instructed—such as the visibility of a child button control.

To set the current visual state, use the static GoToState method of the VisualStateManager class (a native Silverlight type). The GroupName property of the TemplateVisualState groups like states together, whereas the Name property of the TemplateVisualState specifies the individual state.

Return to Expression Blend. In the myPlayer template, click myPlayer directly above the designer window, then click Edit Template | Edit Current. Click the States tab and scroll down SuggestionStates as shown in **Figure 9**.

The two SuggestionStates created by the attributes appear as Hide and Show. If you click on Hide, a red circle appears just to the left, indicating Expression Blend is recording any property changes made within the designer. Expression Blend continues to record property changes until Hide is clicked again, which causes the red recording circle to disappear.

With Expression Blend actively recording for the Hide visual state, set the buttons to Collapsed. Select all three buttons under the Objects and Timeline window and choose Collapsed as their Visibility in the Properties tab. Stop recording for the Hide visual state by clicking the Hide button once again. Now click Show so that a red circle appears to the left of the Show visual state. This time explicitly record Visible as the visibility status by clicking the Advanced Property Options button just to the right of the Visibility drop-down and selecting Record Current Value. Save all open documents and once again return to Visual Studio.

The native Silverlight class, VisualStateManager, is used to explicitly set a currently active visual state. From within the OnApplyTemplate method of the player, set Hide as the currently active visual state:

```
VisualStateManager.GoToState(this, "Hide", true);
```

Within suggestion_FetchCompleted, set Show as the currently active state to display the buttons once the stream has ended and the Suggestion data download has completed:

```
VisualStateManager.GoToState(this, "Show", true);
```

To hide the buttons once a button is clicked (or the original stream is replayed), create a new event handler for the MediaElement's MediaOpened event, and set the visual state to Hide.

Launch and debug the player one final time. You'll see the buttons are invisible until the very end of the video, at which point they become visible. Clicking a button navigates the player to whatever URL was specified in the button's corresponding Suggestion setting.

The SMF project space on Codeplex gives you access to the code base, documentation, discussions and the issue tracker. Take a look and contribute what you can. The more creative minds applied to the project, the better the result for everyone. ■

BEN RUSH *is an 18-year veteran software developer specializing in the Microsoft .NET Framework and related Microsoft technologies. He enjoys smart code and fast bike rides.*

XCEED

DataGrid

for WPF

Serious recognition

Microsoft®
Visual Studio® Team System 2010

"Using Xceed DataGrid for WPF in Microsoft Visual Studio Team System 2010 helped us greatly reduce the time and resources necessary for developing all the data presentation features we needed. Working with Xceed has been a pleasure."

Norman Guadagno,
Director of Product Marketing
for Microsoft Visual Studio Team System

IBM®
U2 SystemBuilder™

"IBM U2 researched existing third-party solutions and identified Xceed DataGrid for WPF as the most suitable tool. The datagrid's performance and solid foundation take true advantage of WPF and provide the extensibility required for IBM U2 customers to take their applications to the next level in presentation design and styling."

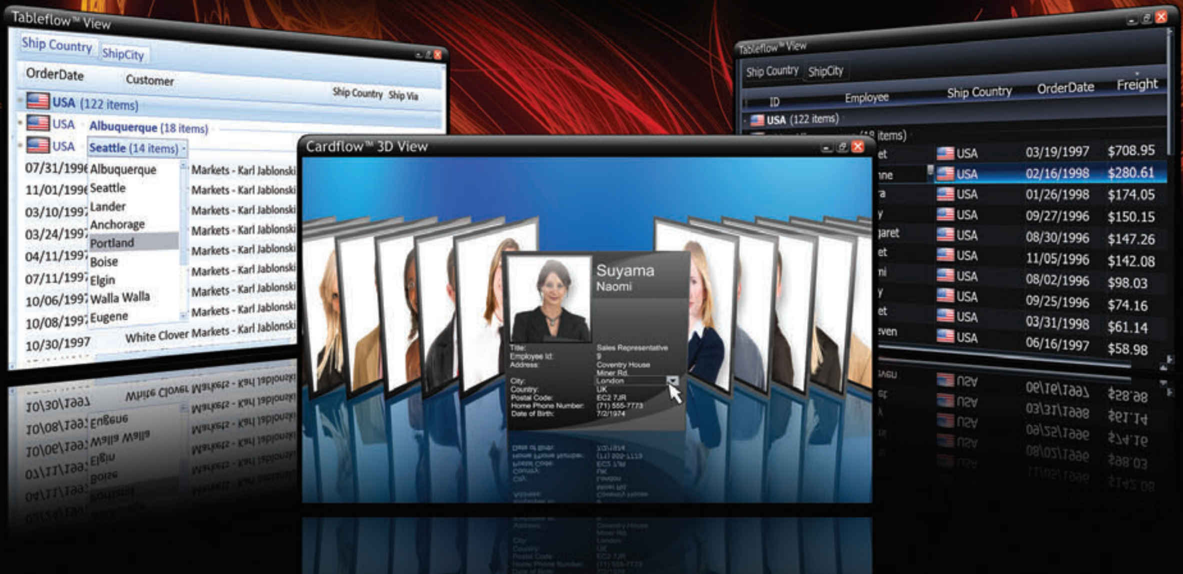
Vincent Smith,
U2 Tools Product Manager at IBM

Microsoft® Visual Studio® PARTNER

XCEED
MULTI-TALENTED COMPONENTS

Datagrids, transformed

- Display & edit data in ***stunning 2D*** or ***3D***
- ***Highest-performance*** WPF datagrid
- Most ***adopted***, most ***mature*** WPF control
- ***150*** features, ***10*** major releases in ***3*** years



The smooth-scrolling Tableflow™ view provides a rich, fluid, and high-performance experience. Its innovative group navigation control redefines datagrid usability.

The Cardflow™ 3D view lets you add a true 3D experience to your application. Also offered is a classic 2D card view.

The first WPF datagrid on the market and under constant development. Build apps you can trust in mission-critical situations.

Try it live on xceed.com

XCEED
MULTI-TALENTED COMPONENTS

Securing Your Silverlight Application

Josh Twist

In my role as a consultant with Microsoft Services, I have regular discussions with customers and partners about application security. In this article, I'll explore some of the themes that arise in those discussions. In particular, I'll focus on the new challenges programmers face when trying to secure Silverlight applications, and I'll consider where development teams should focus their resources.

This article touches on many technical concepts that you'll find covered in more detail elsewhere (including this magazine). For this reason, I won't explore these topics in great technical depth. Instead, the goal of the article is to "connect the dots" and show how you can exploit these concepts to secure your applications.

This article discusses:

- Silverlight topology
- Windows authentication
- Forms authentication
- ASP.NET authorization
- Authorization in WCF Services and WCF Data Services
- Cross-domain authentication
- Securing XAP files

Technologies discussed:

Silverlight, WCF Services, WCF Data Services, Windows Presentation Foundation, ASP.NET

When planning security for an application, it's useful to think of three A's: authentication, authorization and audit.

Authentication is the act of confirming that users are who they claim to be. We usually do this with a user name and password.

Authorization is the process of confirming that a user, once authenticated, actually has the appropriate permissions to perform a particular action or access a particular resource.

Audit is the act of maintaining a record of activity such that actions and requests made upon a system can't be denied by the user.

I will focus on the first two, authentication and authorization, in the context of a Silverlight application. As this is a Rich Internet Application (RIA), the majority of concepts described in this article apply equally to Asynchronous JavaScript and XML (AJAX) or other RIA approaches. I'll also discuss how you can prevent unwanted access to your Silverlight application files.

Topology

Silverlight is a cross-browser plug-in that leverages many of the graphical concepts pioneered by Windows Presentation Foundation (WPF), enabling Web developers to create rich user experiences far beyond what's possible with only HTML and JavaScript.

Unlike ASP.NET, Silverlight is a client-side technology, so it runs on users' computers. So Silverlight development arguably has more in common with Windows Forms or WPF than with ASP.NET. In many ways, this is one of Silverlight's greatest advantages, as it removes many of the problems caused by the stateless nature

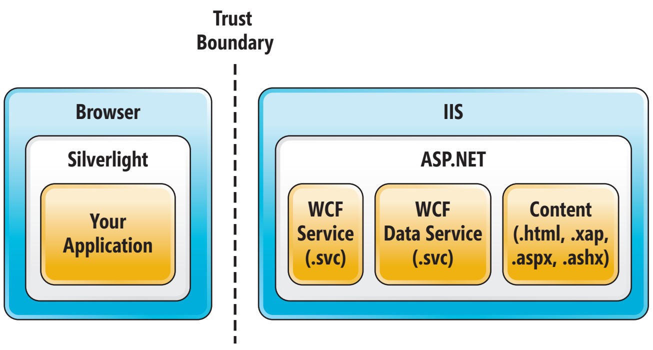


Figure 1 **Silverlight Runs on the Wrong Side of the Trust Boundary**

of Web applications. However, because all the UI code runs on client computers, you can't trust it anymore.

Services

Unlike Windows Forms, Silverlight operates within the browser sandbox and has a reduced set of capabilities, so it provides an increased degree of security (though in Silverlight 4, users can identify certain applications as trusted and promote the programs' privileges to allow COM interop). Because of this, Silverlight can't connect to a database directly, so you must create a layer of services that provide access to your data and business logic.

Typically, you host these services on your Web server, just as you would with your ASP.NET Web forms, for example. Given that Silverlight code runs on the wrong side of the trust boundary between your servers and the real world (see **Figure 1**), the focus of your team's effort should always be to secure the services.

There's little point in implementing rigorous security checks within your Silverlight code itself. After all, it would be easy for an attacker to do away with the Silverlight application altogether and invoke your services directly, side-stepping any security measures you implemented. Alternatively, a malicious person could use a utility like Silverlight Spy or Debugging Tools for Windows to change the behavior of your application at runtime.

This is an important realization—a service can't know for sure what application is invoking it or that the app hasn't been modified in some way. Therefore your services have to ensure that:

- The caller is properly authenticated
- The caller is authorized to perform the requested action

For those reasons, most of this article focuses on how to secure services in a way that's compatible with Silverlight. Specifically, I'll consider two different types of services hosted via ASP.NET in Microsoft IIS. The first type, services created using Windows Communication Foundation (WCF), provides a unified programming model for building services. The second, WCF Data Services (formerly ADO.NET Data Services), builds on WCF to let you rapidly expose data using standard HTTP verbs, an approach known as Representational State Transfer (REST).

Naturally, if security is a concern, it's always wise to encrypt any communication between clients and servers. The use of HTTPS/SSL encryption is recommended and assumed throughout this article.

Today, the two most common authentication methods Web developers use on the Microsoft platform are Windows authentication and forms authentication.

Windows Authentication

Windows authentication leverages the Local Security Authority or Active Directory to validate user credentials. This is a big advantage in many scenarios; it means you can centrally manage users with tools already familiar to systems administrators. Windows authentication can use any scheme supported by IIS including basic, digest, integrated authentication (NTLM/Kerberos) and certificates.

The integrated scheme is the most common choice for use with Windows authentication, because users don't have to provide their user names and passwords a second time. Once a user logs on to Windows, the browser can forward credentials in the form of a token or a handshake that confirms the person's identity. There are some disadvantages to using integrated authentication, because both the client and server need visibility of the user's domain. As a result, it's best targeted at intranet scenarios. Furthermore, though it works with Microsoft Internet Explorer automatically, other browsers, such as Mozilla Firefox, require additional configuration.

Both basic and digest authentication typically require users to re-enter their user names and passwords when they initiate a session with your Web site. But because both are part of the HTTP specification, they work in most browsers and even when accessed from outside your organization.

Silverlight leverages the browser for communication, so Windows authentication is easy to implement with any of the IIS authentication methods just discussed. For a detailed description of how to do so, I recommend reading the step-by-step guide "How to: Use basicHttpBinding with Windows Authentication and TransportCredentialOnly in WCF from Windows Forms" at msdn.microsoft.com/library/cc949012. This example actually uses a Windows Forms test client, but the same approach applies to Silverlight.

Forms Authentication

Forms authentication is a mechanism that provides simple support for custom authentication in ASP.NET. As such, it's specific to HTTP, which means it's also easy to use in Silverlight.

The user enters a user name and password combination, which is submitted to the server for verification. The server checks the credentials against a trusted data source (often a database of users), and if they're correct, returns a FormsAuthentication cookie. The client then presents this cookie with subsequent requests. The cookie is signed and encrypted, so only the server can decrypt it—a malicious user can neither decrypt nor tamper with it.

Exactly how you invoke forms authentication varies depending on how you implement your login screen. For example, if you've used an ASP.NET Web form that redirects to your Silverlight application after the user's credentials have been validated, you probably have no more authentication work to do. The cookie already will have been sent to the browser and your Silverlight application will continue to use the cookie whenever making a request to that domain.

If, however, you want to implement the login screen inside your Silverlight application, you'll need to create a service that exposes your authentication methods and sends the appropriate cookie. Fortunately, ASP.NET already provides what you need—the authentication service. You just need to enable it in your application. For detailed guidance, I recommend reading "How to: Use the ASP.NET Authentication Service to Log In through Silverlight Applications" at msdn.microsoft.com/library/dd560704(VS.96).

Another great feature of ASP.NET authentication is its extensibility. A membership provider describes the mechanism by which the user name and password are verified. Fortunately, there are a number of membership providers available as part of ASP.NET, including one that can use SQL Server databases and another that uses Active Directory. However, if a provider that meets your requirement isn't available, it's straightforward to create a custom implementation.

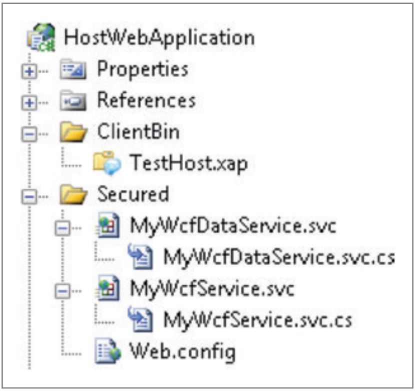


Figure 2 **Secured Folder Containing the Web.config File**

ASP.NET Authorization

Once your users are authenticated, it's important to ensure that only they can attempt to invoke the services. Both ordinary WCF services and WCF Data Services are represented by a .svc file in ASP.NET applications. In this example, the services are going to be hosted via ASP.NET in IIS, and I'll demonstrate how you can use folders to secure access to the services.

Securing .svc files this way is a little confusing because, by default, a request for such a file actually skips most of the ASP.NET pipeline, bypassing the authorization modules. As a result, to be able to rely on many ASP.NET features, you'll have to enable ASP.NET compatibility mode. In any case, the WCF Data Services mandate that you enable it. A simple switch inside your configuration file achieves the task:

```
<system.serviceModel>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
</system.serviceModel>
<system.web>
  <authorization>
    <deny users="?"/>
  </authorization>
</system.web>
```

With ASP.NET compatibility enabled, it's possible to prevent access to unauthenticated users by using the authorization section of a web.config file, also shown in the previous code snippet.

When using forms authentication, the developer must think carefully about which parts of the site need to be accessible, even to unauthenticated users. For example, if all parts are restricted to authenticated users only, how will an unauthenticated user log in?

It's often easiest to create a folder structure that supports your basic authorization requirements. In this example, I've created a "Secured" folder that contains the MyWcfService.svc and MyWcf-DataService.svc files, and I've deployed a web.config file. In **Figure 2** you can see the folder structure, and the previous code snippet shows the contents of the web.config file.

Note that the root of the application must have anonymous access allowed, otherwise users won't be able to reach the login page.

For sites using Windows authentication, things can be somewhat simpler in this respect, as authentication takes place before the user gets to the resources contained within the application, so there's no need for a specific login page. Using this approach, it's actually possible to restrict access to services in a more detailed way, allowing only specific groups of users or roles to access resources. For more information, see "ASP.NET Authorization" (msdn.microsoft.com/library/wce3kxhd).

This example implements authorization somewhat, but folder-level authorization alone is far too coarse-grained to rely on for most scenarios.

Authorization in WCF Services

Using the PrincipalPermission attribute is an easy way to demand that an invoker of a Microsoft .NET Framework method be within a specific role. This code sample demonstrates how this might be applied to a ServiceOperation in WCF where the calling user must be part of the "OrderApprovers" role:

```
[PrincipalPermission(SecurityAction.Demand, Role = "OrderApprovers")]
public void ApproveOrder(int orderId)
{
  OrderManager.ApproveOrder(orderId);
}
```

This is easily implemented in applications that use Windows authentication to leverage the existing facility to create Active Directory groups for organizing users. With applications using forms authentication, it's possible to leverage another great provider-based feature of ASP.NET: RoleProviders. Again, there are a number of these available, but if none are suitable, you can implement your own.

Of course, even per-method authorization is rarely enough to meet all your security needs, and you can always fall back to writing procedural code inside your services as shown in **Figure 3**.

WCF is a highly extensible platform, and as with all things in WCF, there are many approaches to implementing authorization in your services. Dominick Baier and Christian Weyer discussed a number of the possibilities in detail in the October 2008 issue of *MSDN Magazine*. The article, "Authorization in WCF-Based Services" (msdn.microsoft.com/magazine/cc948343), even ventures

Figure 3 **Using Procedural Code to Implement Specific Authorization**

```
Public void CancelOrder(int orderId)
{
  // retrieve order using Entity Framework ObjectContext
  OrdersEntities entities = new OrdersEntities();
  Order orderForProcessing = entities.Orders.Where(o => o.Id ==
    orderId).First();

  if (orderForProcessing.CreatedBy !=
    Thread.CurrentPrincipal.Identity.Name)
  {
    throw new SecurityException(
      "Orders can only be canceled by the user who created them");
  }

  OrderManager.CancelOrder(orderForProcessing);
}
```

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.

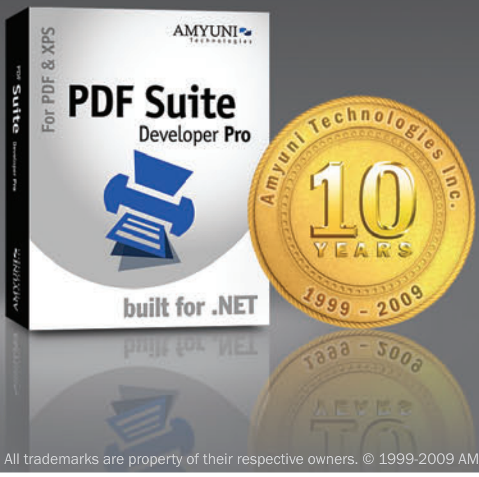
Customization

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

Figure 4 **A WCF Data Services Code-Behind File with Configuration of Entity Set Access Rules**

```
Public class MyWcfDataService : DataService<SalesEntities>
{
  // This method is called only once to initialize service-wide policies.
  Public static void InitializeService(IDataServiceConfiguration config)
  {
    config.SetEntitySetAccessRule("Orders", EntitySetRights.AllRead);
    config.SetEntitySetAccessRule("Products", EntitySetRights.AllRead |
      EntitySetRights.WriteAppend | EntitySetRights.WriteDelete);
  }}
```

into claims-based security, a structured way of organizing the authorization in your application.

Authorization in WCF Data Services

WCF Data Services, as the name suggests, builds on WCF to provide REST-based access to a data source—perhaps most often a LINQ-to-SQL or LINQ-to-Entity Framework data source. In brief, this lets you provide access to your data using a URL that maps to the entity sets exposed by your data source (an entity set typically maps to a table in your database). Permissions to these entity sets can be configured inside the services code-behind file. **Figure 4** shows the content of the MyWcfDataService.svc.cs file.

Here, I've given Read permissions over the Orders entity set and configured the Products entity set to allow full reading, the inserting of new records and the deletion of existing records.

However, because WCF Data Services automatically renders access to your data based on this configuration, you don't have direct access to the code, so there's no obvious place to implement any specific authorization logic. WCF Data Services supports interceptors that allow developers to implement logic between the client and the data source. For example, it's possible to specify a query interceptor that filters the results for a particular entity set. The example in **Figure 5** shows two query interceptors added to the MyWcfDataService class.

The first is applied to the Products entity set and ensures that users can retrieve only products created by them. The second ensures that only users in the PrivateOrders role can read orders flagged Private.

Likewise, it's possible to specify change interceptors that run before an entity is inserted, modified or deleted as demonstrated here:

```
[ChangeInterceptor("Products")]
public void OnChangeProducts(Product product, UpdateOperations operations
{
  if (product.CreatedBy != Thread.CurrentPrincipal.Identity.Name)
  {
    throw new DataServiceException(
      "Only products created by a user can be deleted by that user");
  }
}
```

On initial viewing, the OnChangeProducts change interceptor in this code sample appears to expose a security vulnerability, because the implementation relies on data passed from an external source—specifically the "product" parameter. But when deleting an entity in WCF Data Services, only an entity key is passed from the client to the server. That means the entity itself, in this case the Product, has to be fetched again from the database and therefore can be trusted.

However, in the case of an update to an existing entity (for example, when the operations parameter equals UpdateOperations.Change), the product parameter is the de-serialized entity sent by the client, therefore it can't be trusted. The client application may have been modified to specify the CreatedBy property of this particular product to a malicious user's own identity, thereby elevating the usurper's privileges. That could allow modification of a product by an individual who shouldn't be able to do so. To avoid this, I recommend that you re-fetch the original entity from the trusted data source based on the entity key alone, as shown in **Figure 6**.

Because this implementation relies so much on the CreatedBy property of the Product entity, it's critically important that this is enforced in a reliable way from the moment the data is created. **Figure 6** also shows how this might be achieved by overriding any value passed by the client for an Add operation.

Note that as the example currently stands, handling operations of type UpdateOperations.Change wouldn't be an issue. In **Figure 4**, the service was configured to allow only AllRead, WriteAppend (insert) and WriteDelete actions to occur on the Products entity sets. Therefore, the ChangeInterceptor would never be invoked for a Change operation, as the service would immediately reject any request to modify a Product entity at this endpoint. To enable updates, the call to SetEntitySetAccessRule in **Figure 4** would have to include WriteMerge, WriteReplace or both.

Cross-Domain Authentication

The Silverlight plug-in can make cross-domain HTTP requests. A cross-domain call is an HTTP request made to a domain other than the one from which the Silverlight application was downloaded. The ability to make such calls has traditionally been viewed as a security vulnerability. It would allow a malicious developer to make requests to another site (for example, your online banking site) and automatically forward any cookies associated with that domain. Potentially, this could give the attacker access to another logged-in session within the same browser process.

For this reason, sites have to opt in to allowing cross-domain calls through the deployment of a cross-domain policy file. This is an XML file that describes what types of cross-domain calls are allowed—for example, from what domain to what URLs. For more information, see "Making a Service Available Across Domain Boundaries" (msdn.microsoft.com/library/cc197955(VS.95)).

You should always exercise caution when deciding to expose any sensitive information to cross-domain calls. But if you decide this is a scenario you need to support alongside authentication, it's important to note that cookie-based authentication methods—like

Figure 5 **Query Interceptors in WCF Data Services**

```
[QueryInterceptor("Products")]
Public Expression<Func<Product, bool>> OnQueryProducts()
{
  String userName =ServiceSecurityContext.Current.PrimaryIdentity.Name;
  return product => product.CreatedBy == userName;
}

[QueryInterceptor("Orders")]
Public Expression<Func<Comments, bool>> OnQueryOrders()
{
  bool userInPrivateOrdersRole =
    Thread.CurrentPrincipal.IsInRole("PrivateOrders");
  return order => !order.Private|| userInPowerUserRole;
}
```

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

Figure 6 **A Change Interceptor Preventing Unauthorized Insert, Update and Delete Operations**

```
[ChangeInterceptor("Products")]
Public void OnChangeProducts(Product product, UpdateOperations operations)
{
  if (operations == UpdateOperations.Add)
  {
    product.CreatedBy = Thread.CurrentPrincipal.Identity.Name;
  }
  else if (operations == UpdateOperations.Change)
  {
    Product sourceProduct = this.CurrentDataSource.Products.Where(p =>
      p.Id == product.Id).First();
    if (sourceProduct.CreatedBy != Thread.CurrentPrincipal.Identity.Name)
    {
      throw new DataServiceException(
        "Only records created by a user can be modified by that user");
    }
  }
  else if (operations == UpdateOperations.Delete &&
    product.CreatedBy != Thread.CurrentPrincipal.Identity.Name)
  {
    Throw new DataServiceException(
      "Only records created by a user can be deleted by that user");
  }
}
```

forms authentication described earlier—are no longer suitable. Instead, you could consider leveraging message credentials, where the user name and password are passed to the server and validated with every call. WCF supports this through the TransportWithMessageCredential security mode. For more information, see "How to: Use Message Credentials to Secure a Service for Silverlight Applications" (msdn.microsoft.com/library/dd833059(VS.95)).

Of course, this approach removes ASP.NET from the authentication process altogether, so it's difficult to leverage alongside ASP.NET authorization, discussed earlier.

Securing Your Silverlight XAP files

People concerned about Silverlight security often ask, "How can I protect my XAP files?" Sometimes the motivation behind this query is to protect the intellectual property contained within the code. In this case, you'll need to look at obfuscation to make it more difficult for people to understand your code.

Another common motivation is to prevent malicious users from interrogating the code and understanding how the Silverlight application works—giving them the potential to break into your services.

I usually respond to this with two points. First, although it's possible to restrict the download of your Silverlight application (.xap file) to authenticated and authorized users only, there's no reason to trust these users to be any less malicious than an unauthenticated user. Once the application has been downloaded to the client, there's absolutely nothing to stop users from interrogating the code in an attempt to elevate their own privileges or forward the libraries to somebody else. Obfuscation may make this process a little more difficult, but it isn't good enough to make your application secure.

Second, it's critically important to remember that anybody who can legitimately call services via your Silverlight application can also call those services directly, using an Internet browser and some JavaScript, for example. There's nothing you can do to stop this from happening, so it's paramount that you focus your security efforts on shoring up your services. Do this right and it doesn't matter what a malicious user can garner from your Silverlight application's code. Nonetheless, some people would still like to make sure that only authenticated users can access their .xap files. This is possible, but how easy it is depends on the version of IIS you're using and your chosen authentication method.

If you're using Windows authentication, then you can easily protect your .xap files using IIS Directory Security. If, however, you're using forms authentication, then things get a little more complicated. In this case, it's up to the FormsAuthenticationModule to intercept and verify the cookie accompanying any request and allow or deny access to the requested resource.

Because the FormsAuthenticationModule is an ASP.NET module, the request must pass through the ASP.NET pipeline for this inspection to take place. In IIS6 (Windows Server 2003) and previous versions, requests for .xap files will not, by default, be routed via ASP.NET.

IIS7 (Windows Server 2008), though, introduced the Integrated Pipeline, which allows all requests to be routed through the ASP.NET pipeline. If you can deploy to IIS7 and use an application pool running in Integrated Pipeline mode, then securing your .xap files is no more difficult than securing your .svc files as described earlier in the ASP.NET Authorization section. But if you have to deploy to IIS6 or earlier, you probably have some additional work to do.

One popular approach involves streaming the bytes that make up your .xap file through another extension that the ASP.NET pipeline does handle. The typical way to do this is via an IHttpHandler implementation (in an .ashx file). For more information see "Introduction to HTTP Handlers" (msdn.microsoft.com/library/ms227675(VS.80)).

Another approach is to change the configuration of IIS so that .xap files are routed through the ASP.NET pipeline. However, because this requires a nontrivial change to your IIS configuration, the former approach is more common.

Another issue to consider with forms authentication is the login screen. If, as proposed earlier in this article, you opt for an ASP.NET Web form, then there are no problems. But if you'd prefer the login screen to be authored in Silverlight, you'll need to break the application into parts. One part (the login module) should be available to unauthenticated users, and another (the secured application) should be available to authenticated users only.

You can take two approaches:

1. Have two separate Silverlight applications. The first would contain the login dialog and be in an unsecured area of the site. On successful login, this would then redirect to a page specifying a .xap file in a secure area of your site.
2. Break your application into two or more modules. The initial .xap, located in an unsecured area of your site, would perform the authentication process. If successful, that .xap file would request a subsequent one from a secure area that could be dynamically loaded into the Silverlight application. I recently blogged about how you can do this (thejoyofcode.com/How_to_download_and_crack_a_Xap_in_Silverlight.aspx). ■

JOSH TWIST *is a principal consultant with the Microsoft Application Development Consulting team in the United Kingdom and can be found blogging at thejoyofcode.com.*

THANKS to the following technical experts for reviewing this article: *Zulfiqar Ahmed, Chris Barker and Simon Ince*

Do Wonders

With the same premium tools used by more than 50% of Fortune 100 companies

Aspose provides extensive file format processing capabilities for some of the most popular file formats including:

- DOCX
- XLSX
- PDF
- PPT
- MPP (Project)
- SWF
- ODF
- InfoPath
- BarCode
- Report
- MSG (Outlook)

The ever expanding list of features now includes the ability to Read, Write, Convert, Print, View, and Render for a variety of platforms including .NET, Java, SQL Server, JasperReports and SharePoint. Extensive File Format Processing coupled with unmatched performance, reliability and an award winning customer support makes sure you perform... beyond the rest!

Logon to **www.aspose.com**
and get your free evaluation copy right now!

ASPOSE®
The .NET & Java Component Publisher™

Enterprise Patterns with WCF RIA Services

Michael D. Brown

Two major announcements from PDC09 and Mix10 were the availability of the Silverlight 4 beta and RC, respectively. By the time you read this, the full release to Web of Silverlight 4 will be available for download. Along with extensive printing support, it includes support for elevated permissions, webcams, microphones, toast, clipboard access and more. With its new feature set, Silverlight 4 is poised to go toe-to-toe with Adobe AIR as a multiplatform rich UI framework.

Although all of that does excite me, I'm primarily a business-application developer, and one thing I would love is a simple way to get my business data and logic into a Silverlight application.

One concern with line-of-business Silverlight applications is connecting to data. Nothing prevents you from creating your own Windows Communication Foundation (WCF) service and connecting to it in Silverlight 3, but that leaves a lot to be desired, especially when you consider the myriad ways you can connect to data from ASP.NET or desktop applications. Whereas desktop and Web applications can connect directly to the database via NHibernate, Entity Framework (EF) or raw ADO.NET constructs, Silverlight apps are separated from my data by "the cloud." I call this separation the data chasm.

Crossing this chasm may seem deceptively simple at first. Obviously, it has been done to some degree in a number of existing data-rich Silverlight applications. But what initially appears to be an easy task becomes increasingly complicated as you address more concerns. How do you track changes over the wire or encapsulate business logic in entities that live on both sides of the firewall? How do you keep the details of transmission from leaking into your business concerns?

Third-party tools are emerging to address these concerns, but Microsoft also saw it needed to provide a solution, so it introduced WCF RIA Services (formerly .NET RIA Services), or for brevity, RIA Services. (You'll find a full introduction to RIA Services in "Building a Data-Driven Expense App with Silverlight 3" in the May 2009 edition of *MSDN Magazine* (msdn.microsoft.com/magazine/dd695920). I've been following it since I was first invited into the beta program, providing suggestions to the development team and learning how to leverage the framework within my own applications.

A common question in the RIA Services forums is how RIA Services fit into best practices architecture. I was always impressed with the basic forms-over-data capabilities of RIA Services, but I definitely saw the opportunity to better architect my application so the framework concerns didn't leak into the logic of my application.

Introducing KharaPOS

I've developed an exemplar application, KharaPOS, to provide a tangible example of the concepts I present in this article. It's a point-of-sale (POS) application implemented in Silverlight 4 using RIA Services, Entity Framework and SQL Server 2008. The ultimate goal is to enable the application to be hosted in the Windows Azure platform and SQL Azure, but there's the little problem of Microsoft .NET Framework 4 support (or lack thereof) with the Windows Azure platform.

In the interim, KharaPOS provides a good example of using the .NET Framework 4 to create a real-world application. The project

This article discusses prerelease versions of Silverlight 4 and WCF RIA Services. All information is subject to change.

This article discusses:

- The forms and controls pattern
- The table data gateway pattern
- The Model-View-ViewModel pattern
- The transaction script pattern
- The domain model pattern
- Repository and query objects

Technologies discussed:

Silverlight, WCF RIA Services, Microsoft .NET Framework, ASP.NET, Entity Framework

Code download available at:

KharaPOS.codeplex.com

is hosted via CodePlex at KharaPOS.codeplex.com. You can visit the site to download the code, view documentation and join the discussion surrounding development of the application.

I should note that I borrowed from the book, "Object Models: Strategies, Patterns, and Applications, Second Edition" (Prentice Hall PTR, 1996), by Peter Coad with David North and Mark Mayfield, for the majority of the design and functionality of the KharaPOS application. I'll focus on a single subsystem of the application, catalog management (see **Figure 1**).

Enterprise Patterns

A number of excellent books discuss design patterns for enterprise application development. One book I constantly use as a reference is "Patterns of Enterprise Application Architecture" (Addison-Wesley, 2003) by Martin Fowler. This book and its supplemental Web site (martinfowler.com/eaaCatalog/) provide an excellent summary of helpful software patterns for developing enterprise business applications.

A handful of patterns in Fowler's catalog deal with the presentation and manipulation of data, and interestingly enough, they occupy the same space as RIA Services. Understanding these will give a clearer picture of how RIA Services can be adapted to meet the needs of the simplest to the most complex business applications. I'll discuss the following patterns:

- Forms and controls
- Transaction script
- Domain model
- Application service layer

Let's take a quick tour of these patterns. The first three concern different ways of dealing with the logic surrounding your data. As you progress through them, the logic moves from being scattered throughout the application and repeated as needed to being centralized and focused.

Forms and Controls

The forms-and-controls pattern (or as I refer to it, forms over data) places all of the logic within the UI. At first glance, this seems like a bad idea. But for simple data-entry and master-detail views, it's the simplest and most direct approach to get from UI to database. Many frameworks have intrinsic support for this pattern (Ruby on Rails scaffolding, ASP.NET Dynamic Data and SubSonic are three prime examples), so there's definitely a time and place for what some call an anti-pattern. While many developers relegate the forms-over-data approach to initial prototyping only, there are definite uses for it in final applications.

Regardless of your opinion on its utility, there's no denying the simplicity or approachability of forms over data. It's not called rapid application development (RAD) because it's tedious. WCF RIA Services brings RAD to Silverlight. Leveraging Entity Framework, RIA Services and the Silverlight Designer, it's possible to create a simple forms-over-data editor against a database table in five steps:

1. Create a new Silverlight business application.
2. Add a new Entity Data Model (EDM) to the created Web application (using the wizard to import the database).
3. Add a domain service to the Web application (be sure to build it first so that the EDM is properly discovered) referencing the data model.

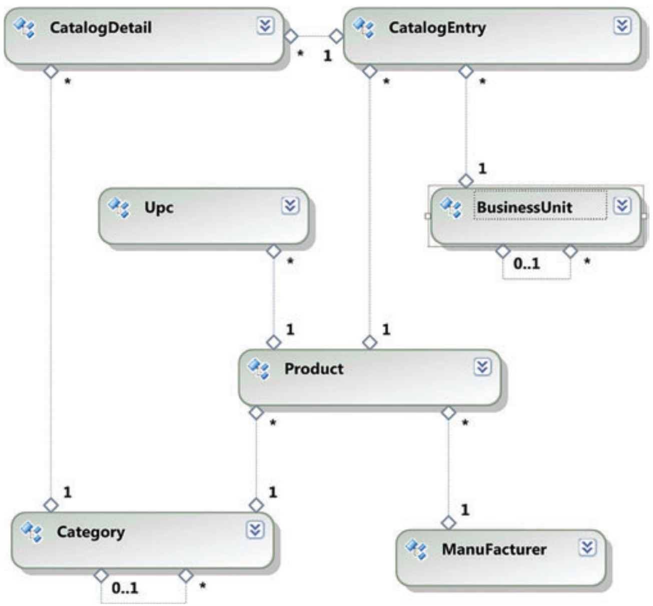


Figure 1 **The Entity Data Model for Catalog Management**

4. Use the data sources panel to drag an entity exposed by RIA Services onto the surface of a page or user control in the Silverlight application (be sure to build again so it can see the new domain service).
5. Add a button and code-behind to save changes on the form to the database with this simple line:

```
this.categoryDomainDataSource.SubmitChanges();
```

You now have a simple data grid that can be used to directly edit existing rows in your table. With a few more additions, you can create a form that lets you add new rows to the table.

Although this pattern has been demonstrated repeatedly, showing the advantage of RAD with WCF RIA Services, it's still relevant here because it provides a baseline for development with the framework. Also, as mentioned, this is a valid pattern within RIA Services-based applications.

Recommendation As with ASP.NET dynamic data, the forms-over-data pattern should be used for simple administration UIs (such as the KharaPOS product category editor), where the logic is simple and straightforward: add, remove and edit rows within a lookup table. But Silverlight and RIA Services scale to much more complex applications, as we'll see now.

Table Data Gateway The standard, out-of-the box approach to RIA Services applications I just discussed can also be viewed as an implementation of the table data gateway pattern as presented on pp. 144–151 of Fowler's book. Through two levels of indirection (EF mapping over the database followed by domain service mapping over EF), I've created a simple gateway to the database tables using basic create, read, update and delete (CRUD) operations returning strongly typed data transfer objects (DTOs).

Technically, this doesn't qualify as a pure table data gateway because of its dual layers of indirection. But if you squint, it closely resembles the table data gateway pattern. To be honest, it would have been a more logical progression to discuss the mapping between the RIA Services and the table data gateway pattern, because all the remaining patterns in the list are data interface patterns, but forms over data is mostly a UI pattern. However, I felt it more prudent to start with the basic scenario and focus on the UI moving back toward the database.

Model-View-ViewModel (MVVM) Even though it's simple to create a functional form using forms over data, there's still some friction involved. **Figure 2**, the XAML for category management, illustrates this.

The column for the parent category in the data grid is a combobox that uses a list of the existing categories so users can select

Figure 2 **XAML for Category Management**

```
<Controls:TabItem Header="Categories">
  <Controls:TabItem.Resources>
    <DataSource:DomainDataSource
      x:Key="LookupSource"
      AutoLoad="True"
      LoadedData="DomainDataSourceLoaded"
      QueryName="GetCategoriesQuery"
      Width="0">
      <DataSource:DomainDataSource.DomainContext>
        <my:CatalogContext />
      </DataSource:DomainDataSource.DomainContext>
    </DataSource:DomainDataSource>
    <DataSource:DomainDataSource
      x:Name="CategoryDomainDataSource"
      AutoLoad="True"
      LoadedData="DomainDataSourceLoaded"
      QueryName="GetCategoriesQuery"
      Width="0">
      <DataSource:DomainDataSource.DomainContext>
        <my:CatalogContext />
      </DataSource:DomainDataSource.DomainContext>
      <DataSource:DomainDataSource.FilterDescriptors>
        <DataSource:FilterDescriptor
          PropertyPath="Id"
          Operator="IsNotEqualTo" Value="3"/>
      </DataSource:DomainDataSource.FilterDescriptors>
    </DataSource:DomainDataSource>
  </Controls:TabItem.Resources>
  <Grid>
    <DataControls:DataGrid
      AutoGenerateColumns="False"
      ItemsSource="{Binding Path=Data,
        Source={StaticResource CategoryDomainDataSource}}"
      x:Name="CategoryDataGrid">
      <DataControls:DataGrid.Columns>
        <DataControls:DataGridTextColumn
          Binding="{Binding Name}" Header="Name" Width="100" />
        <DataControls:DataGridTemplateColumn
          Header="Parent Category" Width="125">
            <DataControls:DataGridTemplateColumn.CellEditingTemplate>
              <DataTemplate>
                <ComboBox
                  IsSynchronizedWithCurrentItem="False"
                  ItemsSource="{Binding Source=
                    {StaticResource LookupSource}, Path=Data}"
                  SelectedValue="{Binding ParentId}"
                  SelectedValuePath="Id"
                  DisplayMemberPath="Name"/>
              </DataTemplate>
            </DataControls:DataGridTemplateColumn.CellEditingTemplate>
            <DataControls:DataGridTemplateColumn.CellTemplate>
              <DataTemplate>
                <TextBlock Text="{Binding Path=Parent.Name}"/>
              </DataTemplate>
            </DataControls:DataGridTemplateColumn.CellTemplate>
          </DataControls:DataGridTemplateColumn>
          <DataControls:DataGridTextColumn
            Binding="{Binding ShortDescription}"
            Header="Short Description" Width="150" />
          <DataControls:DataGridTextColumn
            Binding="{Binding LongDescription}"
            Header="Long Description" Width="*" />
        </DataControls:DataGrid.Columns>
    </DataControls:DataGrid>
  </Grid>
</Controls:TabItem>
```

the parent category by name instead of memorizing the ID of the category. Unfortunately, Silverlight doesn't like it when the same object is loaded twice within the visual tree. Therefore, I had to declare two domain data sources: one for the grid and one for the lookup combobox. Also, the code-behind for managing categories is rather convoluted (see **Figure 3**).

I'm not going to give a full tutorial on MVVM here—see the article, "WPF Apps with the Model-View-ViewModel Design Pattern" in the February 2009 issue (msdn.microsoft.com/magazine/dd419663) for an excellent treatise on the topic. **Figure 4** shows one way to leverage MVVM within an RIA Services application.

As you see, the ViewModel is responsible for initializing the domain context and informing the UI when a load is occurring, along with handling the requests from the UI to create new categories, save changes to existing categories and reload the data from the domain service. This leaves a clean separation between the UI and the logic that drives it. The MVVM pattern may appear to require more work, but its beauty reveals itself the first time you have to change the logic for getting data into the UI. Also, moving the process of loading the categories into the ViewModel allows us to clean up the view significantly (XAML and code-behind alike).

Recommendation Use MVVM to prevent complex UI logic from cluttering your UI—or worse, from cluttering your business object model.

Transaction Script

As you begin adding logic to your application, the forms-over-data pattern becomes cumbersome. Because the logic regarding what can be done with the data is embedded in the UI (or in the ViewModel, if you've taken that step), it will be scattered across the application. Another side effect of decentralized logic is that developers may not be aware that specific functionality already exists in the application, which can lead to duplication. This creates nightmares when the logic changes, because it needs to be updated in all locations (assuming that all locations implementing the logic have been properly cataloged).

The transaction script pattern (pp. 110–115 in Fowler's book) provides some relief. It allows you to separate the business logic that manages the data from the UI.

As defined by Fowler, the transaction script "organizes business logic by procedures where each procedure handles a single request from the presentation." Transaction scripts are much more than simple CRUD operations. In fact, they sit in front of the table data gateway to handle CRUD operations. Taken to the extreme, a separate transaction script would handle every retrieval and submission to the database. But, being logical people, we know there's a time and place for everything.

A transaction script is useful when your application has to coordinate an interaction between two entities, such as when you create an association between two instances of different entity classes. For example, in the catalog management system, I signify that a product is available for a business unit to order for its inventory by creating a catalog entry. The entry identifies the product, the business unit, a product SKU and the time during which it can be ordered both internally and externally. To simplify the creation of catalog entries, I created a method on the domain service (see the following code snippet) that provides a transaction script for modifying product availability for a business unit without the UI having to manipulate catalog entries directly.

In fact, catalog entries aren't even exposed through the domain service, as shown here:

```
public void CatalogProductForBusinessUnit(Product product, int businessUnitId)
{
  var entry = ObjectContext.CreateObject<CatalogEntry>();
  entry.BusinessUnitId = businessUnitId;
  entry.ProductId = product.Id;
  entry.DateAdded = DateTime.Now;
  ObjectContext.CatalogEntries.AddObject(entry);
  ObjectContext.SaveChanges();
}
```

Rather than being exposed as a function on the client-side domain context, RIA Services generates a function on the entity in question (in this case Product) that when called, places a change notification on the object that on the server side gets interpreted as a call to the method on the domain service.

Fowler recommends two approaches for implementing the transaction script:

1. With commands that encapsulate the operations and can be passed around
2. With a single class that holds a collection of transaction scripts

I took the second approach here, but there's nothing preventing you from using commands. The benefit of not exposing the catalog entry to the UI layer is that the transaction script becomes the only means of creating catalog entries. If you use the command pattern, the rule is enforced by convention. If a developer forgets a command exists, you'll end up right back where you started with logic fragmentation and duplication.

Another benefit of placing the transaction script on the domain service is that the logic executes server side (as I mentioned earlier). If you have proprietary algorithms or want to be certain the user hasn't manipulated your data maliciously, placing the transaction script on the domain service is the way to go.

Recommendation Use the transaction script when your business logic becomes too complex for forms over data, you want to execute the logic for an operation on the server side, or both.

Business Logic vs. UI Logic I make several references to UI logic versus business logic, and though the difference may seem subtle at first, it's important. UI logic is the logic concerned with the presentation—what's shown on screen and how (for example, the items used to populate a combobox). Business logic, on the other hand, is what drives the application itself (for example, the discount applied to an online purchase). Both are important facets of an application, and another pattern emerges when they're allowed to mix—see the article, "Big Ball of Mud," by Brian Foote and Joseph Yoder (laputan.org/mud).

Passing Multiple Entities to the Domain Service By default, you can pass only one entity to a custom domain service method. For example, the method

```
public void CatalogProductForBusinessUnit(Product product, int businessUnitId)
```

will not work if you attempt to use this signature instead of an integer:

```
public void CatalogProductForBusinessUnit(Product product, BusinessUnit bu)
```

RIA Services would not generate a client proxy for this function because … well, those are the rules. You can have only one entity in a custom service method. This shouldn't pose a problem in most situations, because if you have an entity, you have its key and can retrieve it on the back end again.

Figure 3 **Code-Behind for Managing Categories**

```
private void DomainDataSourceLoaded(object sender, LoadedDataEventArgs e)
{
  if (e.HasError)
  {
    MessageBox.Show(e.Error.ToString(), "Load Error", MessageBoxButton.OK);
    e.MarkErrorAsHandled();
  }
}

private void SaveButtonClick(object sender, RoutedEventArgs e)
{
  CategoryDomainDataSource.SubmitChanges();
}

private void CancelButtonClick(object sender, RoutedEventArgs e)
{
  CategoryDomainDataSource.Load();
}

void ReloadChanges(object sender, SubmittedChangesEventArgs e)
{
  CategoryDomainDataSource.Load();
}
```

Let's just say, for the sake of demonstration, that it's an expensive operation to retrieve an entity (perhaps it's on the other side of a Web service). It's possible to tell the domain service that you want it to hold a copy of a given entity, as shown here:

```
public void StoreBusinessUnit(BusinessUnit bu)
{
  HttpContext.Current.Session[bu.GetType().FullName+bu.Id] = bu;
}

public void CatalogProductForBusinessUnit(Product product, int businessUnitId)
{
  var currentBu = (BusinessUnit)HttpContext.Current.
    Session[typeof(BusinessUnit).FullName + businessUnitId];
  // Use the retrieved BusinessUnit Here.
}
```

Because the domain service is running under ASP.NET, it has full access to the ASP.NET session and the cache, as well, in case you want to automatically remove the object from memory after a certain period of time. I'm actually using this technique on a project where I have to retrieve customer relationship management (CRM) data from multiple remote Web services and present it to the user under a unified UI. I use an explicit method because some data is worth caching and some isn't.

Domain Model

Sometimes business logic becomes so complex that even transaction scripts can't properly manage it. Frequently, this shows up as complex branching logic within a transaction script or multiple transaction scripts to account for nuances in the logic. Another sign that an application has outgrown the utility of transaction scripts is the need for frequent updates to address rapidly changing business requirements.

If you've noticed any of those symptoms, it's time to consider a rich domain model (pp. 116–124 in the Fowler book). The patterns covered so far have one thing in common: The entities are little more than DTOs—they contain no logic (this is considered by some to be an anti-pattern referred to as Anemic Domain Model). One of

Figure 4 **Category Management Through a View Model**

```
public CategoryManagementViewModel()
{
  _dataContext = new CatalogContext();
  LoadCategories();
}

private void LoadCategories()
{
  IsLoading = true;
  var loadOperation= _dataContext.Load(_dataContext.
    GetCategoriesQuery());
  loadOperation.Completed += FinishedLoading;
}

protected bool IsLoading
{
  get { return _IsLoading; }
  set
  {
    _IsLoading = value;
    NotifyPropertyChanged("IsLoading");
  }
}

private void NotifyPropertyChanged(string propertyName)
{
  if (PropertyChanged!=null)
    PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}

void FinishedLoading(object sender, EventArgs e)
{
  IsLoading = false;
  AvailableCategories=
    new ObservableCollection<Category>(_dataContext.Categories);
}

public ObservableCollection<Category>AvailableCategories
{
  get
  {
    return _AvailableCategories;
  }
  set
  {
    _AvailableCategories = value;
    NotifyPropertyChanged("AvailableCategories");
  }
}
```

the major benefits of object-oriented development is the ability to encapsulate data and the logic associated with it. A rich domain model takes advantage of that benefit by putting the logic back into the entity where it belongs.

The details of designing a domain model are beyond the scope of this article. See the book, "Domain-Driven Design: Tackling Complexity in the Heart of Software" (Addison-Wesley, 2004), by Eric Evans, or the previously mentioned Coad book on object models for great coverage on this topic. I can, though, provide a scenario that helps illustrate how a domain model can manage some of this stress.

Some KharaPOS customers want to look at historical sales of certain lines and decide, on a market-by-market basis, whether the lines will be expanded (making more products from it available), reduced, cut all together or remain flat for a given season.

I already have the sales data in another subsystem of KharaPOS, and everything else I need is here in the catalog system. I'll just bring a read-only view of product sales into our entity data model as shown in **Figure 5**.

Now all I have to do is add the product-selection logic to the domain model. Because I'm selecting products for a market, I'll put the logic on the BusinessUnit class (use a partial class with a shared.cs or shared.vb extension to inform RIA Services you want this function to be shuttled to the client). **Figure 6** shows the code.

Performing an auto-select for products to carry over a season is as simple as calling the new function on BusinessUnit and following that by a call to the SubmitChanges function on DomainContext. In the future, if a bug is found in the logic or the logic needs to be updated, I know exactly where to look. Not only have I centralized the logic, I've also made the object model more expressive of the intent. On p. 246 of his "Domain-Driven Design" book, Evans explains why this is a good thing:

If a developer must consider the implementation of a component in order to use it, the value of encapsulation is lost. If someone other than the original developer must infer the purpose of an object or operation based on its implementation, that new developer may infer a purpose that the operation or class fulfills only by chance. If that was not the intent, the code may work for the moment, but the conceptual basis of the design will have been corrupted, and the two developers will be working at cross-purposes.

To paraphrase, by explicitly naming the function for its purpose and encapsulating the logic (along with a few comments to make it clear what's happening), I've made it easy for the next guy (even if the next guy is me five months from now) to determine what's happening before I even go to the implementation. Putting this logic with the data to which it's naturally associated takes advantage of the expressive nature of object-oriented languages.

Recommendation Use a domain model when your logic is complex and gnarly and may involve several entities at once. Bundle the logic with the object to which it has the most affinity and provide a meaningful, intentional name for the operation.

The Difference Between Domain Model and Transaction Script in RIA Services You may have noticed that for both the transaction script and the domain model, the call was made directly on the entity. Note, though, the logic for the two patterns lies in two separate places. In the case of the transaction script, calling

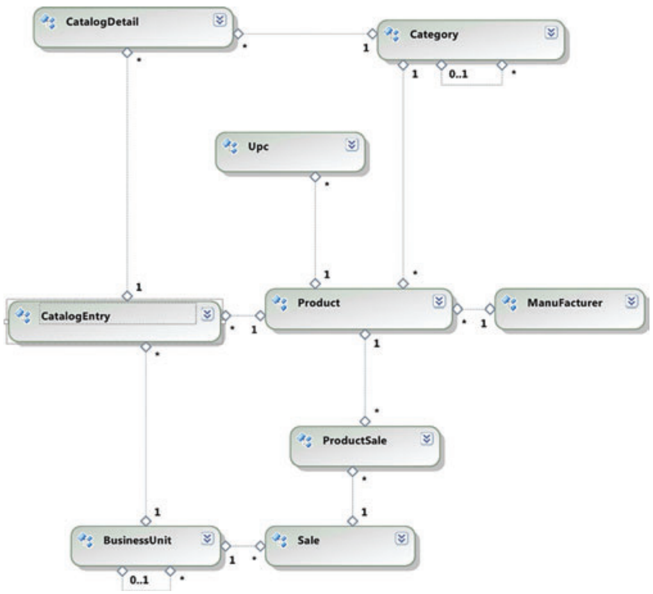


Figure 5 **Entity Data Model Updated with Sales Data**

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 6 **Domain Logic for Selecting Products for a Business Unit**

```
public partial class BusinessUnit
{
  public void SelectSeasonalProductsForBusinessUnit(
    DateTime seasonStart, DateTime seasonEnd)
  {
    // Get the total sales for the season
    var totalSales = (from sale in Sales
                      where sale.DateOfSale > seasonStart
                      && sale.DateOfSale < seasonEnd
                      select sale.LineItems.Sum(line => line.Cost)).
                      Sum(total=>total);
    // Get the manufacturers for the business unit
    var manufacturers =
      Catalogs.Select(c =>c.Product.ManuFacturer).
        Distinct(new Equality<ManuFacturer>(i => i.Id));
    // Group the sales by manufacturer
    var salesByManufacturer =
      (from sale in Sales
      where sale.DateOfSale > seasonStart
      && sale.DateOfSale < seasonEnd
      from lineitem in sale.LineItems
      join manufacturer in manufacturers on
      lineitem.Product.ManufacturerId equals manuFacturer.Id
      select new
      {
        Manfacturer = manuFacturer,
          Amount = lineitem.Cost
      }).GroupBy(i => i.Manfacturer);
    foreach (var group in salesByManufacturer)
    {
      var manufacturer = group.Key;
      var pct = group.Sum(t => t.Amount)/totalSales;
      SelectCatalogItemsBasedOnPercentage(manufacturer, pct);
    }
  }

  private void SelectCatalogItemsBasedOnPercentage(
    ManuFacturer manufacturer, decimal pct)
  {
     // Rest of logic here.
  }
}
```

the function on the entity just serves to indicate to the domain context/service that the corresponding function should be called on the domain service the next time submit changes is called. In the case of the domain model, the logic is executed client side and then committed when submit changes is called.

The Repository and Query Objects The domain service naturally implements the repository pattern (see Fowler, p. 322). In the WCF RIA Services Code Gallery (code.msdn.microsoft.com/RiaServices), the RIA Services team provides an excellent example of creating an explicit implementation of the pattern over the DomainContext. This allows for increased testability of your application without the need to actually hit the service layer and the database. Also, in my blog (azurecoding.net/blogs/brownie) I provide an implementation of the query object pattern (Fowler, p. 316) over the repository, which defers server-side execution of the query until actual enumeration.

Application Service Layer

Quick question: What do you do when you want to leverage a rich domain model but don't want to expose its logic to the UI layer? That's where the application service layer pattern (Fowler, p. 133) comes in handy. Once you have the domain model, this pattern is simple to implement by moving the domain logic out of shared.cs into a separate partial class and placing a function on the domain service that invokes the function on the entity.

The application service layer acts as a simplified facade over your domain model, exposing operations but not their details. Another benefit is that your domain objects will be able to take internal dependencies without requiring the service layer clients to take them as well. In some cases (see the seasonal product-selection example shown in **Figure 6**), the domain service makes one simple call on your domain. Sometimes it might orchestrate a few entities, but be careful—too much orchestration turns it back into a transaction script, and the benefits of encapsulating the logic within the domain are lost.

Recommendation Use the application service layer to provide a simple facade over the domain model and remove the requirement the UI layer needs to take dependencies your entities might take.

Extra Credit: The Bounded Context

In the RIA forums, participants often ask, "How do I break up my very large database across domain services so that it's more manageable?" A follow-up question is, "How do I handle entities that need to exist in multiple domain services?" Initially I thought there shouldn't be a need to do these things; the domain service should act as a service layer over your domain model and a single domain service should serve as a facade over your entire domain.

During my research for this article, though, I came across the bounded-context pattern (Evans, p. 336), which I'd read about before but hadn't remembered when I originally responded to those questions. The basic premise of the pattern is that on any large project there will be multiple sub-domains in play. Take for example KharaPOS, where I have a domain for catalogs and a separate one for sales.

The bounded context allows those domains to coexist peacefully even though there are elements shared between them (such as Sale, Business Unit, Product and LineItem, which are in both the sales and catalog domains). Different rules apply to the entities based on which domain is interacting with them (Sale and LineItem are read-only in the catalog domain). A final rule is that operations never cross over contexts. This is makes life convenient, because Silverlight doesn't support transactions over multiple domain services.

Recommendation Use bounded contexts to divide a large system into logical subsystems.

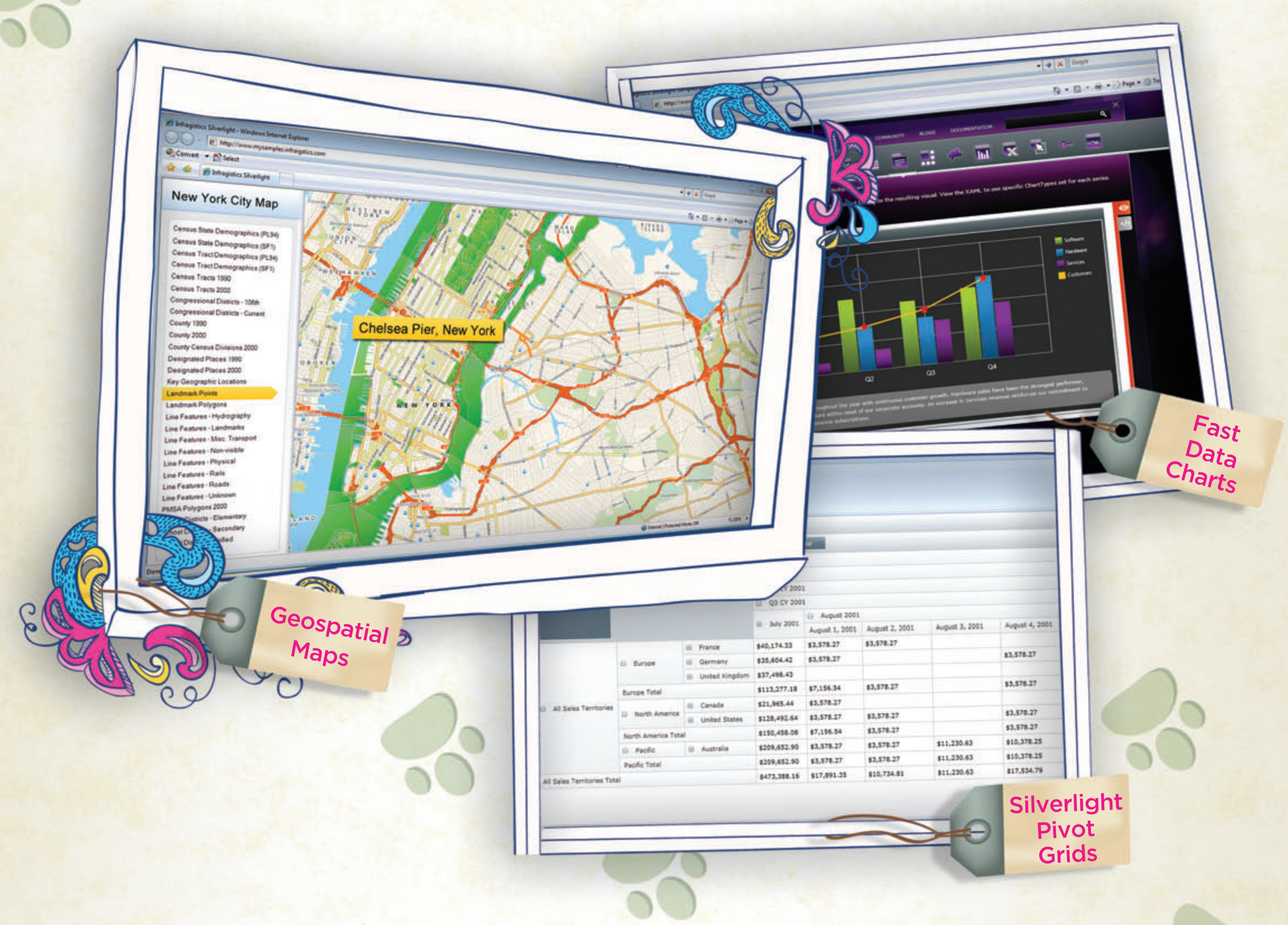
The Pit of Success

In this article, we've seen how RIA Services support the major enterprise patterns with very little effort. Rarely are frameworks so approachable while also being flexible enough to support everything from the simplest spreadsheet data-entry applications to the most complex business applications without requiring major effort to make the transition. This is the "Pit of Success" Brad Abrams mentioned in his blog posting,of the same name (blogs.msdn.com/brada/archive/2003/10/02/50420.aspx). ■

MIKE BROWN *is the president and cofounder of KharaSoft Inc. (kharasoft.com), a technology firm specializing in training, custom software development and Software as a Service. He is an accomplished technology specialist with more than 14 years of industry experience, a back-to-back MVP Award recipient, cofounder of the Indy Alt.NET user group (indyalt.net) and die-hard Bears fan!*

THANKS to the following technical expert for reviewing this article: *Brad Abrams*

INSPIRE! EMPOWER! IMPRESS!



			CY 2001				
			Q3 CY 2001				
				August 2001			
			July 2001	August 1, 2001	August 2, 2001	August 3, 2001	August 4, 2001
All Sales Territories	Europe	France	$40,174.33	$3,578.27	$3,578.27		
		Germany	$35,604.42	$3,578.27			$3,578.27
		United Kingdom	$37,498.43				
	Europe Total		$113,277.18	$7,156.54	$3,578.27		$3,578.27
	North America	Canada	$21,965.44	$3,578.27			
		United States	$128,492.64	$3,578.27	$3,578.27		$3,578.27
	North America Total		$150,458.08	$7,156.54	$3,578.27		$3,578.27
	Pacific	Australia	$209,652.90	$3,578.27	$3,578.27	$11,230.63	$10,378.25
	Pacific Total		$209,652.90	$3,578.27	$3,578.27	$11,230.63	$10,378.25
All Sales Territories Total			$473,388.16	$17,891.35	$10,734.81	$11,230.63	$17,534.79

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

Visual Design of Workflows with WCF and WF 4

Leon Welicki

Developers are increasingly adopting service-oriented architecture (SOA) as a way of building distributed applications. For the uninitiated, designing and implementing service-oriented distributed apps can be intimidating. However, the Microsoft .NET Framework 4 makes it easier than ever to implement Windows Communication Foundation (WCF) services using Windows Workflow Foundation (WF).

WCF workflow services provide a productive environment for authoring long-running, durable operations or services where enforcing an application protocol via ordering of operations is important. Workflow services are implemented using WF activities that can make use of WCF for sending and receiving data.

In this article, I will explain how to combine several features of WCF and WF that were introduced in the .NET Framework 4 to model a long-running, durable and instrumented mortgage-approval process for a real estate company, without having to write code. This article is not intended to be a general introduction to WCF or WF, nor does it walk you through the entire process of creating a working solution. Instead, I'm going to focus on the use of important new .NET Framework 4 features through a practical business scenario. A full working solution is included in the code download for this article.

This article discusses:

- Describing processes in Flowcharts
- Branching workflows
- Asynchronous processes and persistence
- Deploying and consuming the service

Technologies discussed:

Windows Communication Foundation, Windows Workflow Foundation

Code download available at:

code.msdn.microsoft.com/mag201005WCF

The Scenario

Let's start by outlining the scenario around which the workflow application was built. Contoso Housing is a real estate company that sells houses and condos. To provide better customer service and an end-to-end buying experience, Contoso partners with three mortgage companies that assist potential customers with their mortgage needs. Each mortgage company offers different interest rates. Contoso prioritizes mortgage vendors by their interest rates to ensure that customers get the best deal (using the assumption that a better rate makes the house more likely to sell).

Customers provide their mortgage request data via a Web application. Each customer enters a customer ID, the price of the house, the amount of down payment, the loan period in years, salary information and some background verification information.

The first step in the application workflow (see **Figure 1**) uses the customer-entered data to screen the customer and determine eligibility prior to sending the request on to the mortgage vendors.

The application follows these rules:

- If the applicant had a foreclosure, a bankruptcy or is in a lawsuit, the petition will be rejected.
- If the applicant does not have credit history and provides a down payment less than 20 percent, the application is returned for revision (incorrect petition), but is not rejected. The customer must provide at least a 20 percent down payment to continue.
- If none of the above apply, the petition is approved.

The mortgage vendors are contacted according to Contoso's preferred order to petition the mortgage request. If a mortgage vendor rejects the applicant, the next one is asked. There's a standard service contract for mortgage approval requests that all vendors implement. During this stage in the process, customers should be able to query the state of their request.

Once the mortgage petition is resolved (one vendor accepted or all rejected), the customer interaction is recorded in Contoso's customer relationship management (CRM) system through a service exposed by the CRM system. The result is returned to the customer.

Note that the mortgage approval process shown in **Figure 1** provides a general description of the business process, but does not have any indication about how to implement it. The workflow service will be an implementation of this process.

Creating a Declarative Service

The workflow service will receive data from the customer, do the initial screening, coordinate the conversation with the mortgage vendors, register the interaction in the CRM service and expose the result to the customer.

The service will be long-running (it can take days or months to complete), durable (it can save the state and resume later), and instrumented (both developers and users will be able to know what's going on without having to debug the service). By using WCF and WF, you can achieve all this declaratively without writing any code—you just have to assemble and configure components provided by the .NET Framework. **Figure 2** shows an architectural diagram of the solution.

A WCF workflow service is contained in a .xamlx file that defines the workflow. You can define business processes visually using the WF designer, and the service contract is inferred based on the structure of the workflow.

To create a workflow service in Visual Studio 2010, create a new project, then select the WCF Workflow Service Application template. This project template creates a very simple (but running) workflow with Receive and SendReply activities. This is similar to a class with a method that receives an integer and returns a string. You can extend the workflow by adding more activities to it. These activities might add execution logic or service operations.

To define the service contract, you use the messaging activities provided by WF. The configuration for the service is stored in the web.config file, just like in regular WCF services. If you open web.config, you'll see a very clean configuration file because WCF workflow services take advantage of the service configuration improvements introduced in WCF 4.

Messaging activities combine WF and WCF in a seamless way. They're designed to support message-oriented workflows and provide better integration of messaging into workflows. Messaging activities enable workflows to send data out to other systems (Send, SendReply and SendAndReceiveReply), and to receive data from other systems (Receive, ReceiveReply and ReceiveAndSendReply). They also include activities for working with correlation (InitializeCorrelation, CorrelationScope) and transactions (TransactedReceiveScope).

Receive and Send activities allow modeling the messaging interactions within a workflow. The contract in a service can be defined by configuring the Receive and Send activities in it. Each Receive is exposed as an operation. Each Send activity sends a message to a service. The target service does not have to utilize WCF or even the .NET Framework because you interact with it through standard protocols.

Send and Receive can be configured to use either a message-based or parameter-based (RPC) approach for receiving and sending the actual data. This controls the wire-format in which the data is sent or received.

Describing Processes in Flowcharts

You can model the process using a Sequence, but a quick look at **Figure 1** shows that at some point the process requires looping back

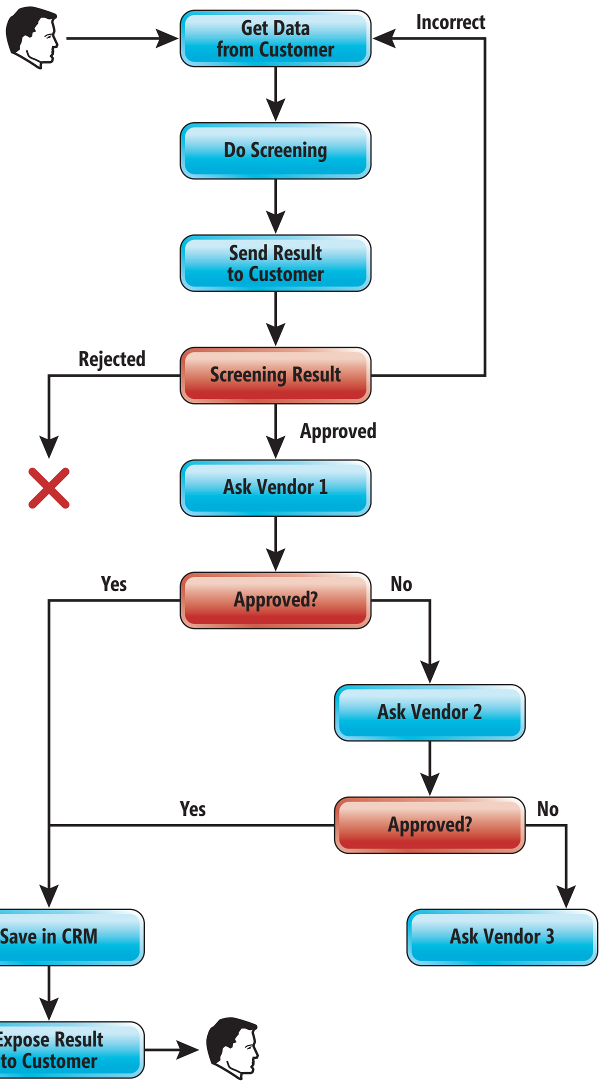


Figure 1 **The Mortgage Approval Process**

to a previous step, and that's not directly supported in sequential workflows. (It would require modeling the loop manually using constructs such as a While activity with a carefully modeled condition.) A Flowchart is a better fit for modeling this scenario.

Flowchart is a new control-flow activity introduced in WF 4 that lets you describe your processes much as you would on a whiteboard. Flowchart describes processes that are sequential in nature, with a single path of execution that may require looping back to a previous step under certain circumstances. To describe the process, the Flowchart uses arrows and boxes, a popular approach among many disciplines.

The default workflow created by the WCF Workflow Service project template is a Sequence, but this does not mean it's the only control-flow activity you can use in workflow services. To use a Flowchart (or any other composite activity) as the root activity in a service, simply delete the Sequence and add a Flowchart.

Figure 3 shows how easily a Flowchart can be created from the diagrams that business users typically draft to describe a process.

Flowchart is a great tool for describing business processes, minimizing the mismatch between executable processes and the way they're specified. In this case, the Flowchart definition *is* the documentation of the process.

But let's slow down a bit. We've jumped from an empty Flowchart to a complete one in no time. Let's go back to the empty Flowchart and start working on it.

The potential home buyer introduces his data through a Web application. This data is passed to the workflow through a Receive activity. The result of including a Receive activity in the Flowchart is that the WorkflowServiceHost exposes an endpoint that allows users to communicate with the workflow by sending messages to it.

As I mentioned earlier, you can configure Send and Receive activities to use either a message-based or RPC-based approach. In this case I configured the Receive activity to use a RPC-based approach. This means it receives a set of input arguments, similar to a method call. To make them available to the workflow, I need to create variables and bind them to the parameters (see **Figure 4**). Another possible approach would be having a DataContract that collapses all these fields.

If you want to create and start a new instance of the workflow upon receiving a message, you need to set the CanCreateInstance property in the Receive activity to true. This means the WorkflowServiceHost will create a new instance when it receives a message from this endpoint.

Modeling the Screening Step with WF Composition

Once you have data in the workflow, you can start working with it. The first step is screening, which means verifying a set of conditions in order to determine whether the applicant is eligible for a mortgage before contacting the vendors.

One approach is to add several decision shapes (FlowDecision) to the main Flowchart. This works, but makes the overall process hard to read. Furthermore, any modifications to the screening rules would require updating the main flow. Flowchart seems to be a good fit for expressing conditions visually, but we want to keep the main process lean.

A solution is to add a new Flowchart inside the existing Flowchart. WF 4 has strong support for composition at its core, so activities can be freely composed. This means you can add a new Flowchart wherever you need it, including inside an existing Flowchart. Moreover, the composition is arbitrary and doesn't impose any limits. You can combine existing activities at your convenience.

The child flowchart is displayed collapsed in the parent (the Flowchart designer does not support expand-in-place). You will need to double-click it to model the screening logic. The screening Flowchart (see **Figure 5**) is a child of the main Flowchart and has access to its variables and arguments.

What if you want to write some code? You could certainly write code to describe the screening process. In this case, for example, you could author a CodeActivity that receives the data from the customer as input, performs the verification (a set of chained if statements in your language of choice), and returns the result. This has its own pro and cons. It offers potentially better performance (all verifications are executed within a single pulse of execution) and more compact representation than the declarative approach. On the other hand, you lose the visual representation of the process (opacity), and changing the process requires modifying the code and recompiling.

Sending Results to the Customer

When the screening verification is complete, I need to return the result to the applicant. I received the applicant data through a Receive activity at the beginning of the workflow. To send the reply back I use a SendReply activity (a SendReply for an existing Receive can be created by right-clicking on the Receive and selecting Create SendReply). The combination of Receive and SendReply allows implementing the request-response message exchange

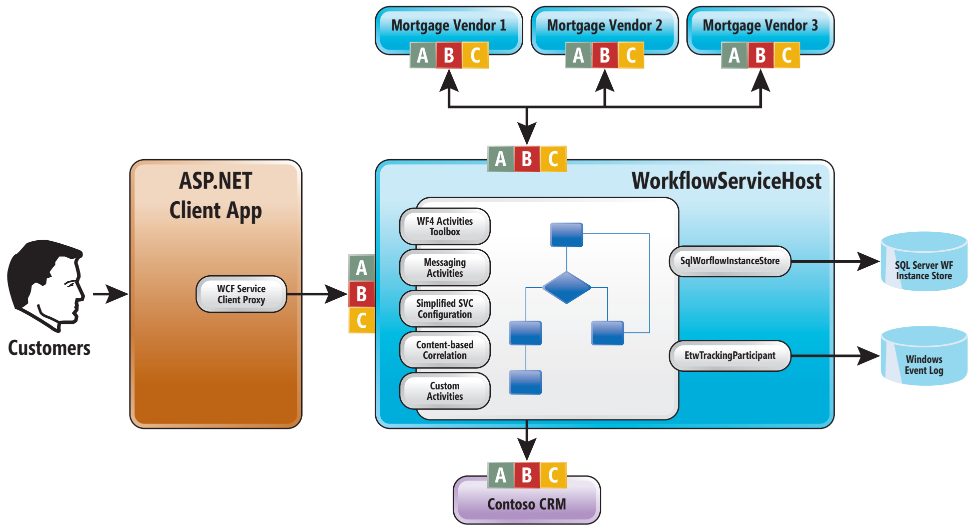


Figure 2 **Architecture of the Solution**

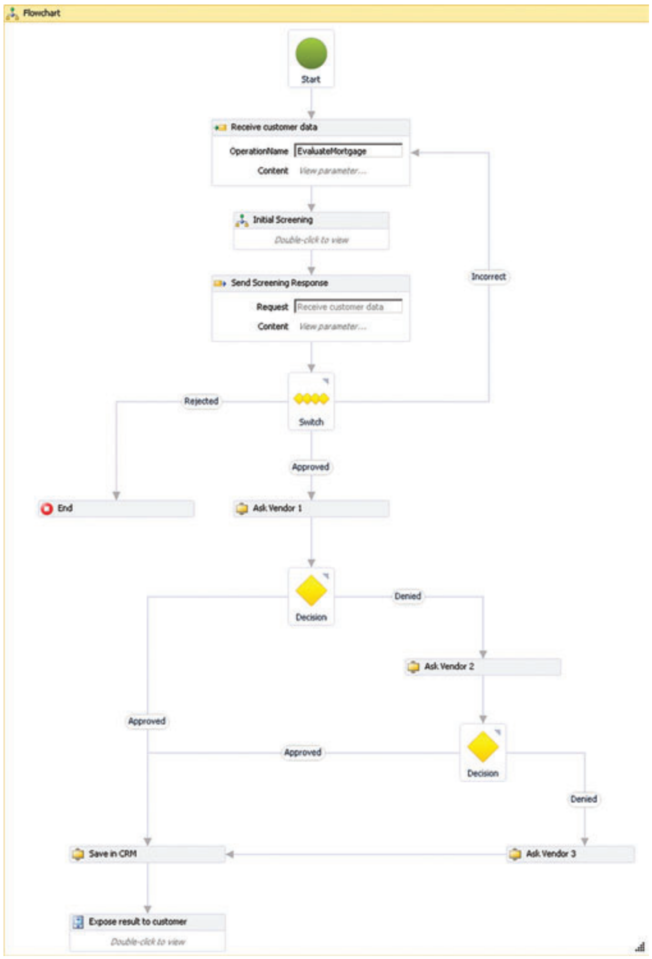


Figure 3 **The Mortgage Approval Process as a Flowchart**

pattern. SendReply is configured to send back the result of the operation and a description message to the customer.

Why SendReply and not Send? You can use a pair of Receive and Send activities to model duplex messaging exchange patterns like "request and wait for a response" (similar to a callback), but SendReply is better suited to modeling request-response message exchange patterns (similar to a method invocation).

Deciding Where to Go Next

When the screening is complete, the workflow can continue with the mortgage petition process. There are two branches at this point: Reject and Approved, but when the applicant needs to provide more data, the workflow must go back to a prior step. Flowchart allows modeling this *drawing-a-line* action to the step where the workflow will proceed.

To decide the path to be taken based on the result of the screening, I use a FlowSwitch activity (similar to a switch statement), as shown in **Figure 6**.

Correlation

When the mortgage request is deemed incorrect, the workflow asks the customer to provide additional data to an existing workflow instance. How can the Receive activity know that the data it's receiving is a correction of the data the customer provided earlier? In other words, how can you send another message to a running instance of a workflow? The answer is correlation.

WF 4 introduces a framework for correlation. A correlation is actually one of two things:

- A way of grouping messages together. A classic example of this is sessions in WCF, or even more simply the relationship between a request message and its reply.
- A way of mapping a piece of data to a workflow service instance.

There are several types of correlation available in the .NET Framework 4. I use content-based correlation in this example workflow. A content-based correlation takes data from the incoming message and maps it to an existing instance. Content-based correlation is used when a workflow service has multiple methods that are accessed by a single client and a piece of data in the exchanged messages identifies the desired instance.

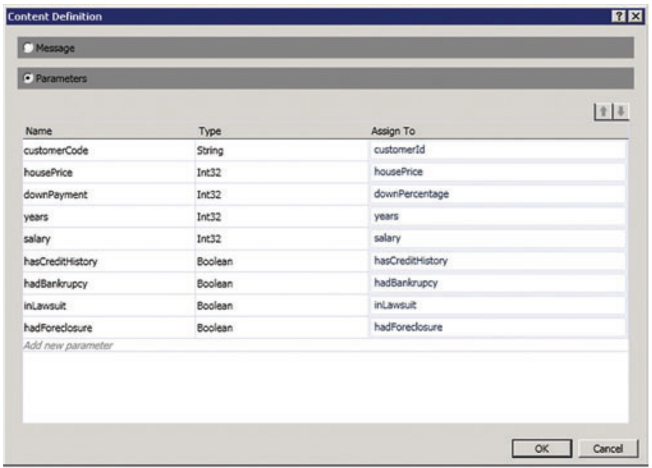
To set up the correlation, I declare a variable of type CorrelationHandle. This is the handle used to store the correlation information (customerCorrelationHandle in the companion solution). The next step is using the correlation handle. The Receive activity's property grid has a Correlations section to configure correlation. It has properties for configuring a correlation handle (CorrelatesWith), for specifying the data that you correlate on through a correlation query (CorrelatesOn) and for initializing a correlation handler (CorrelationInitializers). I configured the CorrelatesWith and CorrelatesOn arguments in the Receive activity to correlate on the customerCode field.

CorrelatesOn and CorrelatesWith can be confusing. Here's a rule that can help you: a Receive CorrelatesWith an existing correlation handle and CorrelatesOn data specified by a correlation query.

All this can be set up using the WF designer.

I want to correlate on the customer identification so I create a correlation query that extracts the customerCode from the message. Note that correlation queries are authored using XPath, but you don't need to know XPath because the WF designer creates the query by inspecting the contract of the received message for you.

The first Receive activity in the workflow is configured to create a new workflow instance upon receiving a message. However, it won't create a new instance when I loop back to it because it's also configured to correlate on the customerCode. Before creating a new instance, it looks for an existing instance with that correlation key.



Name	Type	Assign To
customerCode	String	customerId
housePrice	Int32	housePrice
downPayment	Int32	downPercentage
years	Int32	years
salary	Int32	salary
hasCreditHistory	Boolean	hasCreditHistory
hadBankrupcy	Boolean	hadBankrupcy
inLawsuit	Boolean	inLawsuit
hadForeclosure	Boolean	hadForeclosure
Add new parameter		

Figure 4 **Configuring Input Parameters**

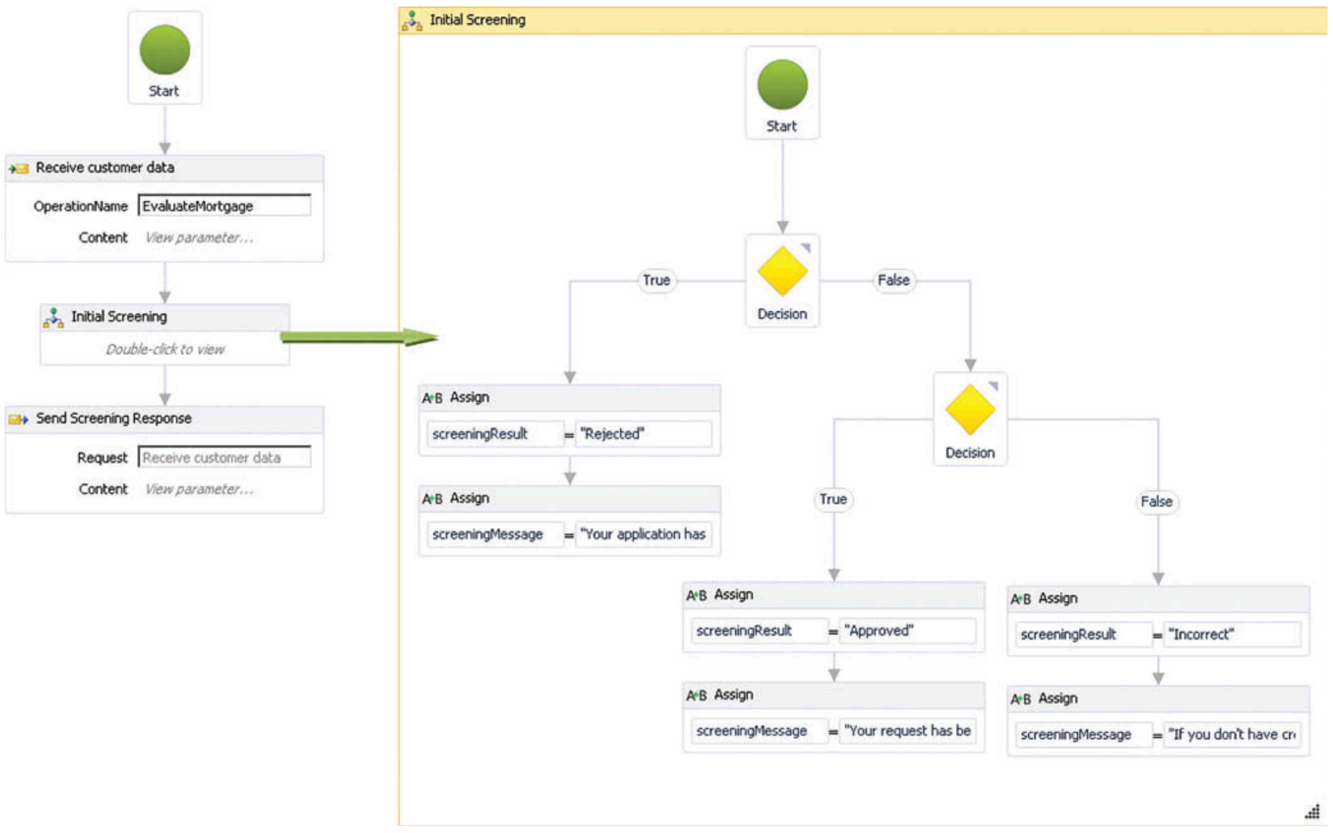

Start
Receive customer data
OperationName EvaluateMortgage
Content View parameter...
Initial Screening
Double-click to view
Send Screening Response
Request Receive customer data
Content View parameter...
Initial Screening
Start
True
Decision
False
Assign
screeningResult = "Rejected"
Assign
screeningMessage = "Your application has
Decision
True
False
Assign
screeningResult = "Approved"
Assign
screeningMessage = "Your request has be
Assign
screeningResult = "Incorrect"
Assign
screeningMessage = "If you don't have cr

Figure 5 **Adding the Screening Flowchart**

Asking Vendors for Rates

If the petition passes the screening, it's sent to the vendors for aluation. Remember that Contoso works with three vendors and has a preferred order for doing business with them. Therefore the process asks vendors in sequential order until one approves.

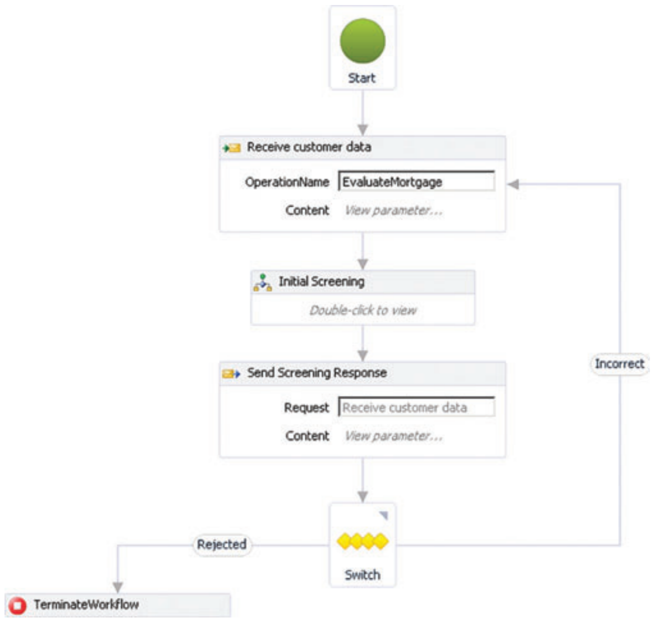


Figure 6 **Each Outbound Arrow in the FlowSwitch Represents a Case in the Switch**

All vendors implement a standard service contract for requesting mortgage approval. The only difference between the interest rate requests to the three mortgage vendors is the URI of the service. However, the vendor interest rate request is complicated by the fact that it involves sending a message and waiting asynchronously for a response while simultaneously responding to customer requests for status.

I could model this process once and copy it three times in the workflow, but this would result in lots of unnecessary duplication and thus a serious maintainability issue. I want to abstract this step so I can use the same process multiple times. Ideally, I would provide the workflow with some input data and get a result when it's done, and that's exactly what custom activities are for.

WF 4 provides two approaches for creating custom activities:

- Declarative create a new activity by composing other existing activities.
- Imperative create a new activity by writing code. There are several base classes with different capabilities that can be derived, including CodeActivity (simple imperative behavior), AsyncCodeActivity (do work asynchronously) and NativeActivity (interact with the WF runtime).

Activities have arguments that define their public signatures in terms of what data they can receive (InArgument) and what data they will return (OutArgument) when they complete. My activity will receive customer mortgage information and a service URI as input, and will return an interest rate and result string message.

I use the Activity item template in Visual Studio to create a new activity declaratively using the WF designer. The designer allows you to author custom activities visually by dragging and dropping existing activities and setting their arguments. The result of authoring a custom activity with the designer is a XAML file that defines an x:Class.

I called the activity AskVendor. The AskVendor activity receives mortgage information from the customer and the URI of the service as input, and provides an interest rate and a result string message as output. If the rate is 0, the petition has been rejected.

AskVendor sends a message to a mortgage vendor asking for an interest rate and waits for a response from the vendor. The response may take minutes, hours or even days to arrive. While waiting for the response, the applicant may want to know the state of the process. Therefore, the activity also responds to status-request messages from the applicant.

To process both actions simultaneously I use a Parallel activity as the root of the custom activity. In one branch I have all the activities for communicating with a mortgage vendor, and in the other the activities to listen for the customer while waiting for the vendor's response. All messaging activities used within AskVendor are configured to correlate on the customerCode field. **Figure 7** shows AskVendor custom activity.

As mentioned, the root of the custom activity is a Parallel. The left branch sends a message to a vendor and waits for a response. Once the response is received, it formats the resulting string message and sets the completed flag to true. The right branch listens for state-query requests from the applicant. The Receive and SendReply are inside of a While activity that's executed until the completed flag is true. The Parallel activity's completion condition (executed when a branch is completed) sets the completed flag to true. Therefore, when the left branch completes (the message from the vendor is received), the completed variable is signaled as true and the While at the right is also completed.

Custom activities are just like any other activity. When you create a custom activity it shows up automatically in the activities toolbox. And using it is no different from using any other existing activity: drag it from the toolbox, drop it in the designer and configure its arguments. Because I haven't created a designer for this activity, the default designer is assigned to it (a rectangle where you can set the DisplayName property). The property grid automatically displays all the arguments of the activity.

I mentioned earlier the strong support for composition in WF 4. This also applies to the custom activities. If you create your own composite activity, you'll be able to freely compose it with any other existing activity like Sequence, Flowchart, Parallel or even other custom activities.

Invoking the CRM service

Contoso's CRM system exposes its core functionality as services. One of these services allows registering an interaction with a customer. I could invoke it using a Send activity, but this would imply configuring the activity manually and importing its service data contracts.

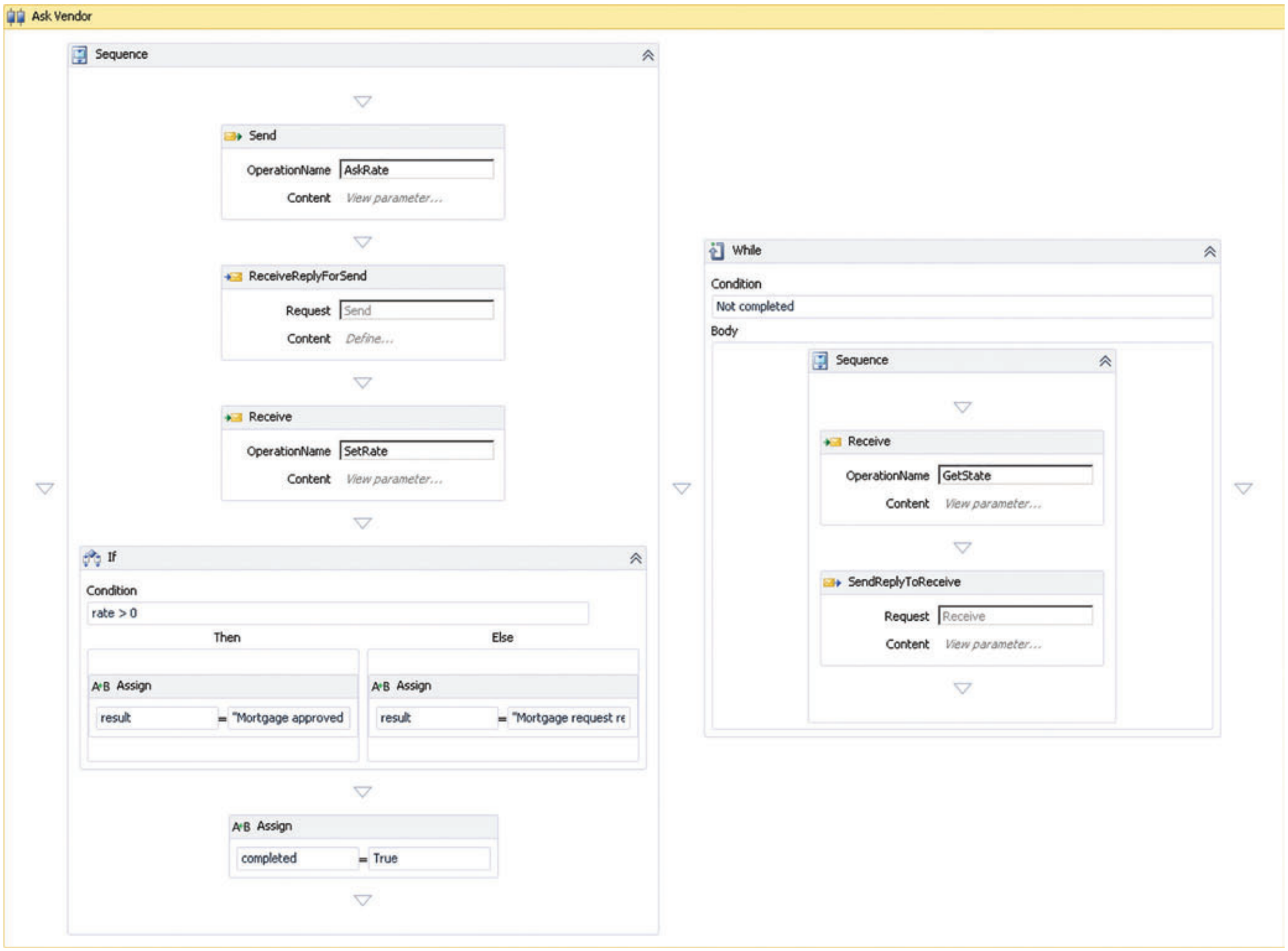
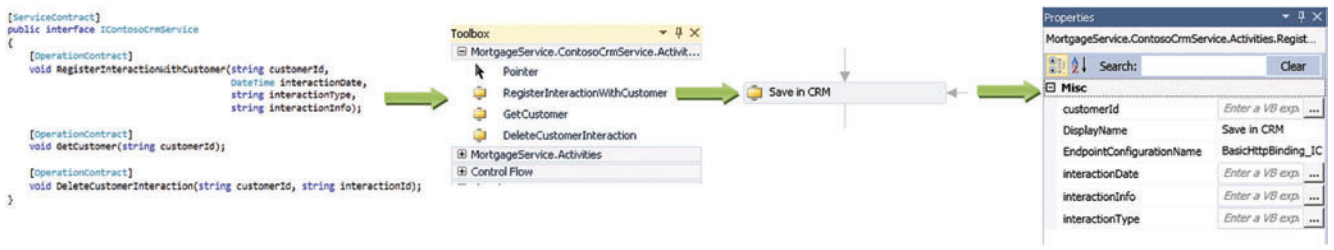


Figure 7 **AskVendor Custom Activity**



```
[ServiceContract]
public interface IContosoCrmService
{
    [OperationContract]
    void RegisterInteractionWithCustomer(string customerId,
                                         DateTime interactionDate,
                                         string interactionType,
                                         string interactionInfo);

    [OperationContract]
    void GetCustomer(string customerId);

    [OperationContract]
    void DeleteCustomerInteraction(string customerId, string interactionId);
}
```

Toolbox
MortgageService.ContosoCrmService.Activit...
Pointer
RegisterInteractionWithCustomer
GetCustomer
DeleteCustomerInteraction
MortgageService.Activities
Control Flow
Save in CRM
Properties
MortgageService.ContosoCrmService.Activities.Regist...
Search:
Clear
Misc

customerId	Enter a VB exp
DisplayName	Save in CRM
EndpointConfigurationName	BasicHttpBinding_IC
interactionDate	Enter a VB exp
interactionInfo	Enter a VB exp
interactionType	Enter a VB exp

Figure 8 **Adding a Service Reference to the CRM Service**

It would be great if I could just import the service into WF and execute it. This is exactly what Add Service Reference in a workflow service project does: given the contract of a service, it automatically creates proxy activities (one for each operation in the service) that can be used to invoke the service (see **Figure 8**). In this case, the CRM service contract has three operations so "Add service reference" has created three activities, which are displayed in the toolbox.

Communicating the Result

Finally, I need to provide the results to the customer. To keep the app simple, I just use a ReceiveAndSendReply activity that exposes the result to the customer. Once the customer reads the result, the workflow completes.

To do this, I need to drop a ReceiveAndSendReply in the Flowchart. Note that when you drop the activity in the designer surface, you get a collapsed Sequence. This is because ReceiveAndSendReply is an activity template, which means that it's a preconfigured set of activities (in this case a Sequence with a Receive and a SendReply). You saw this earlier when I added the child Flowchart for screening.

To configure ReceiveAndSendReply, I need to drill in and set the endpoint information in the Receive activity. I also need to configure Receive to correlate by customer identification so that, when the customer sends a message with his customerCode, he will get a response.

Figure 9 **Specifying the Tracking Participant Profile**

```
<tracking>
  <profiles>
    <!--The health monitoring profile queries for workflow instance level 
records and for workflow activity fault propagation records-->
    <trackingProfile 
      name="HealthMonitoring">
      <workflow activityDefinitionId="*">
        <workflowInstanceQueries>
          <workflowInstanceQuery>
            <states>
              <state name="Started"/>
              <state name="Completed"/>
              <state name="Aborted"/>
              <state name="UnhandledException"/>
            </states>
          </workflowInstanceQuery>
        </workflowInstanceQueries>
        <faultPropagationQueries>
          <faultPropagationQuery 
            faultSourceActivityName ="*" 
            faultHandlerActivityName="*"/>
        </faultPropagationQueries>
      </workflow>
    </trackingProfile>
  </profiles>
</tracking>
```

Long-Running Work

Once the workflow sends an approval request to a vendor, the service instance will be sitting idle waiting for a response. The response can come in minutes, hours, days or even weeks. This poses some interesting challenges. You probably don't want to keep all instances in memory because that would consume unnecessary system resources and wouldn't scale. And if the host process crashes (or in a less apocalyptic scenario, needs to be shut down for maintenance), any unfinished instances would be lost.

Wouldn't it be great if, when an instance is not doing any work, you could just save it to durable storage and remove it from memory? You can—via the WF persistence framework, which allows saving a workflow instance to a storage medium to be retrieved later.

This means instances are not tied to any existing process or machine. In the example workflow, the screening can occur in one process, asking the first vendor for a rate can occur in another, and receiving the response can occur in a third—without affecting the workflow instance's execution or its data. This achieves better use of resources, improving scalability and providing resilience—a crash in the host does not produce the loss of the active instances because they can be resumed from the point at which they were last persisted.

Workflow instances are saved to an instance store. WF 4 includes a SQL Server-based instance store, and the persistence framework is extensible so you can write your own. (The PurchaseProcess sample in the SDK shows how to write a very simple text file instance store, for example.)

I use the built-in SQL Server instance store to persist instances of the Contoso workflow. The good news is that you don't need to write any code to use it. In workflow services, persistence is a behavior that can be configured in the web.config file like so:

```
<!--Set up SQL Instance Store-->
<sqlWorkflowInstanceStore connectionString="Data Source=.\
SQLExpress;Initial Catalog=InstanceStore;Integrated 
Security=True;Asynchronous Processing=True"/>

<!--Set the TimeToUnload to 0 to force the WF to be unloaded. To have a 
durable delay, the workflow needs to be unloaded-->
<workflowIdle timeToUnload="0"/>
```

The first line configures the persistence behavior to use the SQL Server instance store. The second line instructs the persistence framework to persist and unload instances as soon as they become idle (this means that when you execute a Receive and become idle waiting for a response, the workflow instance will be unloaded and saved in the database).

If a workflow is configured to be persistent and has correlation configured, the host (WorkflowServiceHost) is responsible for

LEADTOOLS®

The World Leader in Imaging Development SDKs

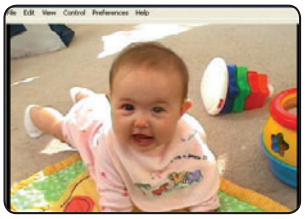
.NET, WPF, WCF, WF, C API, C++ Class Lib, COM & more!

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.**

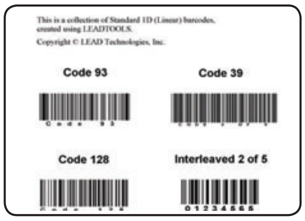
LEADTOOLS provides developers easy access to decades of expertise in color, grayscale, document, medical, vector and multmedia imaging development. Install LEADTOOLS to eliminate months of research and programming time while maintaining high levels of quality, performance and functionality.

•**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

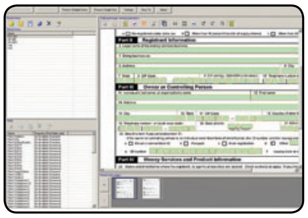


Barcode


This is a collection of Standard 1D (Linear) barcodes, created using LEADTOOLS.
Copyright © LEAD Technologies, Inc.

Code 93 | Code 39
Code 128 | Interleaved 2 of 5

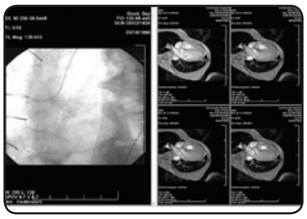
Form Recognition & Processing



Mark-up

Document

DICOM Medical



High Level Design • Low Level Control

LEAD TECHNOLOGIES INCORPORATED

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

Figure 10 **Consuming the Service**

```
protected void OnSubmit(object sender, EventArgs e) {
  using (ContosoRealEstate.ContosoRealEstateClient client =
    new ContosoRealEstate.ContosoRealEstateClient()) {

    string message = "";
    string result = client.EvaluateMortgage(
      out message,
      this.txtId.Text,
      Convert.ToInt32(this.txtHousePrice.Text),
      Convert.ToInt32(this.txtDownpayment.Text),
      Convert.ToInt32(this.txtYears.Text),
      Convert.ToInt32(this.txtSalary.Text),
      this.chkCreditHistory.Checked,
      this.chkBankrupcy.Checked,
      this.chkLawsuit.Checked,
      this.chkForeclosure.Checked);
    lblMessage.CssClass= result;
    lblMessage.Text = message + " (" + result + ")";

    this.btnSubmit.Visible = result.Equals("Incorrect");
    this.btnMonitor.Visible = result.Equals("Approved");
  }
}
```

loading the correct instance when a message arrives based on the correlation information.

Suppose you have an instance that's configured to correlate on customerCode, say for customerCode = 43. You ask a mortgage vendor for a rate and the instance is persisted while waiting for the response (persistence includes saving correlation information). When the mortgage company sends back a message for customerCode = 43, WorkflowServiceHost automatically loads that instance from the instance store and dispatches the message.

Note that the SQL instance store is not installed by default. You need to explicitly install it by running a set of scripts provided with the .NET Framework 4.

Tracking the Service

So I have a long-running service that communicates with other services in a message-based fashion. The service is configured to be durable, does some of its work in parallel, and can execute asynchronous activities. This seems pretty complex. What if something goes wrong? How can you tell which activity failed? What if you just want to know what is going on with an instance of the service?

Visual Studio allows debugging workflows (and you can do step debugging in the WF designer by setting breakpoints, just as you do in code), but this is not an option in production environments.

Instead, WF includes a rich tracking infrastructure that provides data about running workflows. Tracking tells you about things that happen in the workflow (tracking events) to a tracking participant that saves these events. Tracking profiles allow filtering the events that a tracking participant receives so it can get just the information it needs.

WF 4 provides a tracking participant that saves data in the Windows Event Log (EtwTrackingParticipant). You can create your own tracking participants by extending TrackingParticipant. For this workflow I used the default EtwTrackingParticipant. You don't need to write any code to use it; just provide proper configuration in the web.config file. I start by configuring the service to use the EtwTrackingParticipant:

```
<!--Set up ETW tracking -->
<etwTracking profileName="HealthMonitoring "/>
```

I also set it to use a HealthMonitoring profile that provides events that will help to assess the health of our service (see **Figure 9**). Now the service provides information on events that help monitor its health and fix problems as they appear. The SDK supplies several samples that show how to create your own tracking participant and how to write a troubleshooting profile.

Deploying and Consuming the Service

So I've created a workflow using the designer and configured it to use persistence and tracking. The only remaining task is to host and run it.

While developing, you can host the service with the built-in Web service host in Visual Studio 2010. To do this, you just need to run the project with the service and you are done.

Hosting the service in a production environment is only slightly more complex. You can host WCF workflow services in IIS or AppFabric application server extensions. With Visual Studio 2010, you can create a package that can be directly imported to IIS. If you decide to use AppFabric, you'll be able to take advantage of features like the dashboard, which provides summarized information about instances of your service, and query the recorded tracking.

The final step is to actually use the service. In this scenario, Contoso wanted a Web-based interface to allow users to interact with the service. This means consuming the service from an ASP.NET application.

The example WCF workflow service you've seen here is just like any other WCF service you can write in plain code. To consume it, you need to add a service reference in the client project. This service reference creates the client proxies to invoke the service.

With the reference in hand you can invoke the service. The client for this service has one operation for each Receive activity in the service. **Figure 10** shows the code used to request a mortgage approval (and therefore start a new instance of the service).

Closing Notes

As you've seen, the .NET Framework 4 provides a rich feature set that can be used to build complex real-world solutions by assembling existing components. The framework also provides extensibility points to tailor these components to specific needs that accommodate a wide variety of scenarios.

WCF workflow services let you describe a long-running, durable, instrumented process declaratively by simply composing existing activities and configuring the service—though you can also write your own code if necessary.

In this article, I combined several features in WF 4 and WCF 4 to build a service that does some work on its own and coordinates conversations with other existing services. I built the entire service without writing code, including a new artifact (a custom activity).

There's a lot more you can do with these .NET Framework 4 technologies. ■

LEON WELICKI *is a program manager in the WF team at Microsoft focusing on Activities and the WF runtime. Prior to joining Microsoft, he worked as lead architect and dev manager for a large Spanish telecom company and as external associate professor on the graduate computer science faculty at the Pontifical University of Salamanca at Madrid.*

THANKS to the following technical experts for reviewing this article: *Dave Cliffe, Vikram Desai, Kavita Kamani and Bob Schmidt*

Does your Team do more than just track bugs?

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

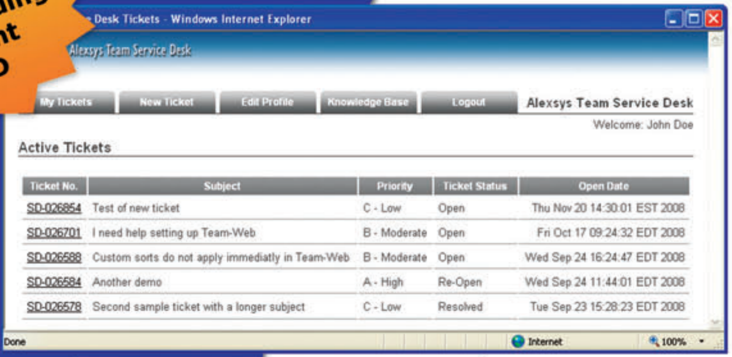
Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

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

Alexsys Team


Desk Tickets - Windows Internet Explorer

Alexsys Team Service Desk

Alexsys Team Service Desk

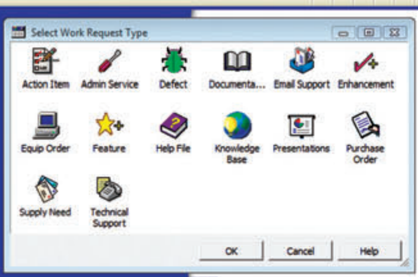
Welcome: John Doe

Active Tickets

Ticket No.	Subject	Priority	Ticket Status	Open Date
SD-026854	Test of new ticket	C - Low	Open	Thu Nov 20 14:30:01 EST 2008
SD-026701	I need help setting up Team-Web	B - Moderate	Open	Fri Oct 17 09:24:32 EDT 2008
SD-026588	Custom sorts do not apply immediatly in Team-Web	B - Moderate	Open	Wed Sep 24 16:24:47 EDT 2008
SD-026584	Another demo	A - High	Re-Open	Wed Sep 24 11:44:01 EDT 2008
SD-026578	Second sample ticket with a longer subject	C - Low	Resolved	Tue Sep 23 15:28:23 EDT 2008

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



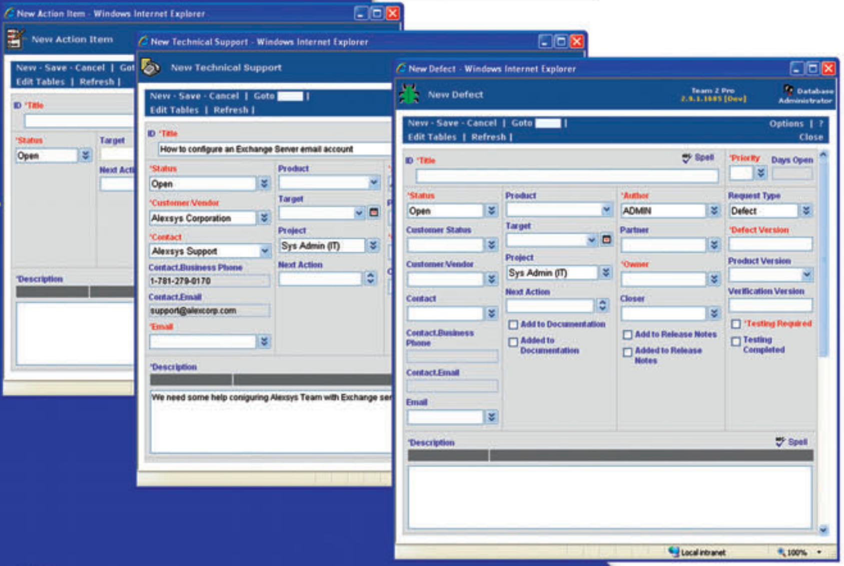
Service Desk Features

- 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


New Action Item - Windows Internet Explorer

Status: Open

New Technical Support - Windows Internet Explorer

Title: How to configure an Exchange Server email account

Status: Open

Customer/Vendor: Alexsys Corporation

Contact: Alexsys Support

Contact.Business Phone: 1-781-279-0170

Contact.Email: support@alexcorp.com

Project: Sys Admin (IT)

Description: We need some help coniguring Alexsys Team with Exchange ser

New Defect - Windows Internet Explorer

Team 2 Pro 2.9.1.1685 [Dev]

Database Administrator

Status: Open

Author: ADMIN

Request Type: Defect

Project: Sys Admin (IT)

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.

JAMES MCCAFFREY

Combinations and Permutations with F#

Understanding combinations and permutations is a fundamental skill in software testing. In this month's Test Run column I show you how to work with combinations and permutations using code written in the new F# language.

A mathematical combination is a subset of k items selected from a set of n items, where order does not matter. For example, if n = 5 and k = 2, all possible ways to select two items from five items are:

```
{0,1}, {0,2}, {0,3}, {0,4}, {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}
```

Notice that I do not list the combination {1,0} because it is considered the same as {0,1}. Also, I have listed the 10 combinations of five items selected two at a time using what is called lexicographical order, where the values in each combination element are listed in increasing order.

A mathematical permutation is all possible rearrangements of n items. For example, if n = 4, all possible permutations listed in lexicographical order are:

```
{0,1,2,3}, {0,1,3,2}, {0,2,1,3}, {0,2,3,1}, {0,3,1,2}, {0,3,2,1},
{1,0,2,3}, {1,0,3,2}, {1,2,0,3}, {1,2,3,0}, {1,3,0,2}, {1,3,2,0},
{2,0,1,3}, {2,0,3,1}, {2,1,0,3}, {2,1,3,0}, {2,3,0,1}, {2,3,1,0},
{3,0,1,2}, {3,0,2,1}, {3,1,0,2}, {3,1,2,0}, {3,2,0,1}, {3,2,1,0}
```

When working with combinations and permutations, two important functions are Choose(n,k) and Factorial(n). You are probably familiar with the Factorial(n) function, which is often abbreviated as n! The Factorial(n) function returns the total number of permutations of order n. For example:

```
4! = 4 * 3 * 2 * 1 = 24.
```

The Choose(n,k) function returns the total number of combinations of k items selected from n items.

```
Choose(n,k) = n! / (k! * (n-k)!)
```

For example:

```
Choose(5,2) = 5! / (2! * (5-2)!) = 5! / (2! * 3!) = 120 / 12 = 10.
```

Combinations and permutations are part of an area of study usually called combinatorial mathematics, or just combinatorics for short.

A good way for you to see where I'm headed in this month's column is to take a look at the screenshot in **Figure 1**. I used Windows PowerShell to host my F# demo application, but I could have just as easily used a command shell. I modified my Windows PowerShell startup script to automatically navigate to the location of my CombinatoricsDemo.exe program.

Behind the scenes, the demo program references and calls into an F# code library named CombinatoricsLib. The demo begins by listing all mathematical combinations of five items selected three at a time. Next, it uses a helper method to apply the last combination element {2,3,4}, to an array of strings {ant, bat, cow, dog, elk} to yield {cow, dog, elk}—that is, the three strings from the original set that are located at index values 2, 3 and 4.

My demo program continues by computing and displaying the value of Choose(200,10), the number of ways to select 10 items from a set of 200 items where order does not matter.

Next, the demo code computes and displays the value of Factorial(52), which is the total number of ways to arrange the cards in a standard deck of 52 cards. Notice that the result is a very, very large number. As I'll explain, F# has the ability to work with arbitrarily large integer values.

My demo program concludes by listing all mathematical permutations of order n = 3.

In the sections that follow, I describe in detail the F# code in the CombinatoricsLib module and the code in the F# demo program shown running in **Figure 1**. Along the way, I compare the use of F# with other languages such as Visual Basic and C# for working with combinations and permutations.

This column assumes you have beginner-to-intermediate experience with a .NET language such as C#, and a very basic familiarity with F#. But even if you are completely new to F#, you should be able to follow my explanations without too much difficulty.

The F# CombinatoricsLib Library

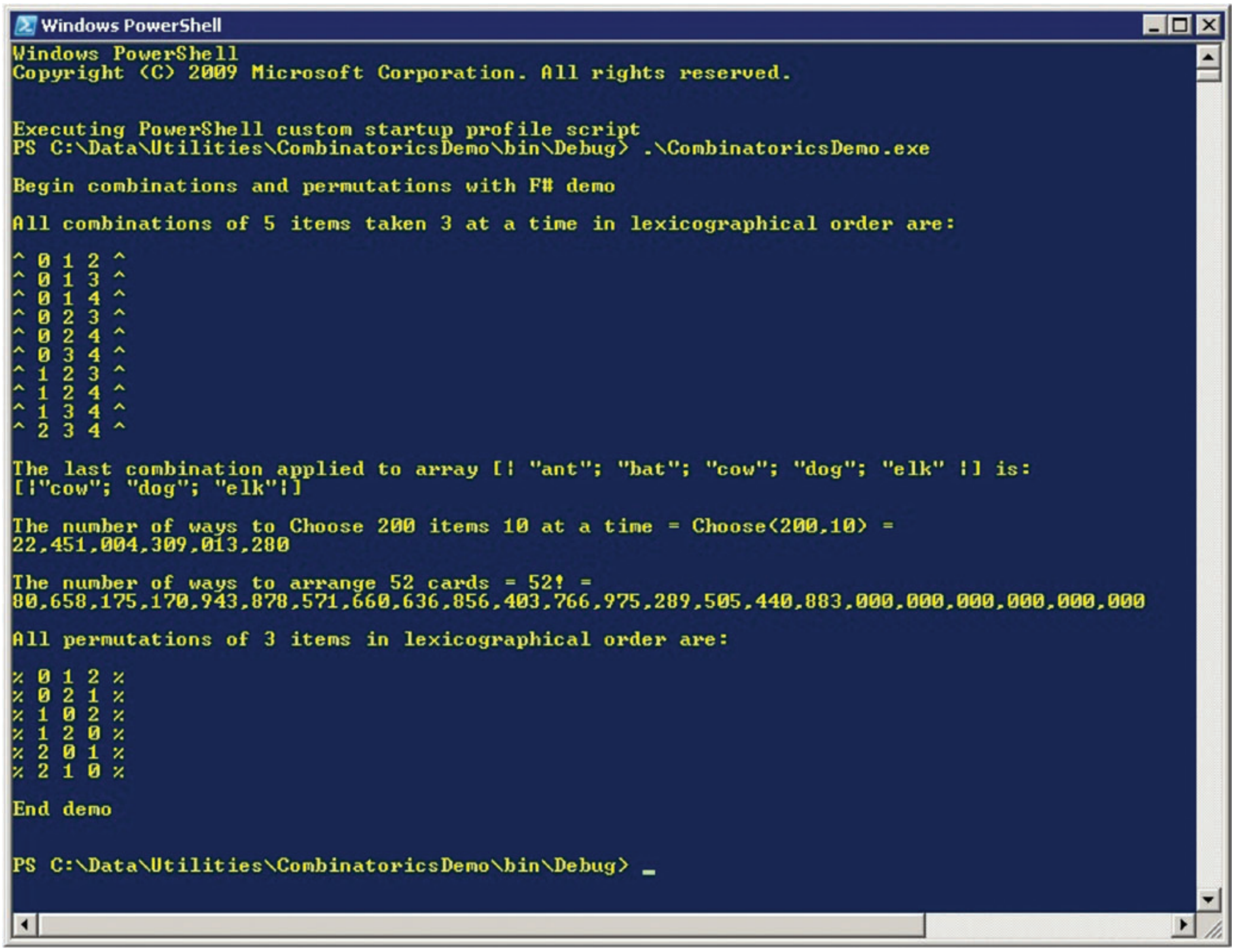
To create my combinatorics library, I used the beta 2 release of Visual Studio 2010, which has the F# language and tools built in. I expect the F# code I present here to work with the release version of Visual Studio 2010 without any significant changes. If you are using an earlier version of Visual Studio, you can find the F# tools on the Microsoft F# Developer Center (msdn.microsoft.com/fsharp). In addition to the F# language, Visual Studio 2010 ships with the Microsoft .NET Framework 4, which my library uses.

I created my library by launching Visual Studio 2010 and selecting File | New | Project. On the new project dialog, I selected the F# Library template and named my library CombinatoricsLib. The overall structure of my CombinatoricsLib is listed in **Figure 2**.

The library code begins with Visual Studio-generated code that names my library Module1. I used this rather nondescript default name instead of changing to something more descriptive so that the syntax for accessing the library will stand out later in this article.

This article discusses a prerelease version of Visual Studio 2010. All information is subject to change.

Code download available at code.msdn.microsoft.com/mag201005TestRun.



```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.


Executing PowerShell custom startup profile script
PS C:\Data\Utilities\CombinatoricsDemo\bin\Debug> .\CombinatoricsDemo.exe

Begin combinations and permutations with F# demo

All combinations of 5 items taken 3 at a time in lexicographical order are:

^ 0 1 2 ^
^ 0 1 3 ^
^ 0 1 4 ^
^ 0 2 3 ^
^ 0 2 4 ^
^ 0 3 4 ^
^ 1 2 3 ^
^ 1 2 4 ^
^ 1 3 4 ^
^ 2 3 4 ^

The last combination applied to array [| "ant"; "bat"; "cow"; "dog"; "elk" |] is:
[|"cow"; "dog"; "elk"|]

The number of ways to Choose 200 items 10 at a time = Choose(200,10) =
22,451,004,309,013,280

The number of ways to arrange 52 cards = 52! =
80,658,175,170,943,878,571,660,636,856,403,766,975,289,505,440,883,000,000,000,000,000,000

All permutations of 3 items in lexicographical order are:

% 0 1 2 %
% 0 2 1 %
% 1 0 2 %
% 1 2 0 %
% 2 0 1 %
% 2 1 0 %

End demo

PS C:\Data\Utilities\CombinatoricsDemo\bin\Debug> _
```

Figure 1 **Combinations and Permutations with F#**

Next, I added two F# open statements to the top-level System namespace and the new System.Numerics namespace so I can access the classes in these namespaces without fully qualifying the class names. The System.Numerics namespace is part of the .NET Framework 4 and contains a BigInteger definition that allows me to work with arbitrarily large integer values.

Defining a type in F# is different from defining a conceptually equivalent C# class. The type definition has a signature that contains the input arguments for the primary type constructor:

```
type Combination(n : int, k : int, a : int[]) =
```

This signature means, "I am defining a type named Combination that has a primary constructor that accepts int arguments n (the total number of items), and k (the subset size), and an integer array a that specifies the individual combination values, such as {0,1,4}."

F# types may have optional secondary constructors, such as the one listed in **Figure 2**:

```
new(n : int, k : int) =
```

Notice that secondary constructors use the explicit new keyword as opposed to type primary constructors. This secondary constructor accepts values just for n and k. With other programming languages such as C#, you would likely define the simpler constructor as the primary constructor, that is, before constructors that accept arguments. But as you'll see in F#, for types with multiple constructors, it is usually better to create a type in such a way that the primary constructor is the one with the most parameters. The structure of my Combination type continues with three member functions:

```
member this.IsLast() : bool =
  ...
member this.Successor() : Combination =
  ...
member this.ApplyTo(a : string[]) : string[] =
  ...
```

The this keyword binds member methods that are publicly visible to external calling code. The IsLast function returns true if the associated Combination object is the last element in lexicographical order, such as {2,3,4} for n = 5 and k = 3, as shown in **Figure 1**.

The Successor function returns the next Combination element in lexicographical order to the current element.

The ApplyTo function accepts an array of strings and returns an array of strings that corresponds to the current Combination element.

My next type member function provides a way to display a Combination element:

```
override this.ToString() : string =
```

I use the override keyword to distinguish my custom ToString function from the base ToString method. Because F# is a .NET language, all objects inherit from a common base Object that has a ToString method. Notice that even though the overridden ToString function has public scope, I do not use the member keyword.

The last two member functions in the Combination type define the Choose and Factorial functions:

```
static member Choose(n : int, k : int) : BigInteger =  ...
static member Factorial(n : int) : BigInteger =  ...
```

Both functions are static, which means that the functions are associated with and called directly from the context of the Combination type rather than a particular instance of a Combination object. Both functions return type BigInteger, defined in namespace System.Numerics, which is directly visible by default to F# code.

In addition to a Combination type, I define a Permutation type as shown in the listing in **Figure 2**.

The F# Combinatorics Library Implementation

Now let's go over the details of the implementation of the CombinatoricsLib library. The Combination primary constructor begins:

```
type Combination(n : int, k : int, a : int[]) =
  do if n < 0 || k < 0 then failwith
    "Negative argument in Combination ctor"
  do if n < k then failwith
    "Subset size k is larger than n in Combination"
```

Interestingly, to perform input argument validation in a primary constructor you should use the do keyword, which indicates an action, rather than the default of a value, which is typically assumed by functional languages such as F#. The failwith keyword throws an exception, which can be caught by calling code.

Figure 2 **F# CombinatoricsLib Structure**

```
module Module1

open System
open System.Numerics // BigInteger class

type Combination(n : int, k : int, a : int[]) =
  // primary constructor code
  new(n : int, k : int) =
    ...
  member this.IsLast() : bool =
    ...
  member this.Successor() : Combination =
    ...
  member this.ApplyTo(a : string[]) : string[] =
    ...
  override this.ToString() : string =
    ...
  static member Choose(n : int, k : int) : BigInteger =
    ...
  static member Factorial(n : int) : BigInteger =
    ...
// end type Combination

type Permutation(n : int, a : int[]) =
  // primary constructor code
  new(n : int) =
    ...
  override this.ToString() : string =
    ...
  member this.Successor() : Permutation =
    ...
  member this.ApplyTo(a : string[]) : string[] =
    ...
  member this.IsLast() : bool =
// end type Permutation
```

Next, I set up the Combination type private members:

```
let n : int = n // private
let k : int = k
let data =
  [| for i = 0 to a.Length-1 do yield a.[i] |]
```

The let keyword binds values to private members. Notice that I can use lowercase n and k as both input parameters and as private fields. This looks a bit awkward and so in most situations I use uppercase notation for either the parameters or the private members.

Copying the values from the input argument array a into the type field data uses a standard F# idiom. The [| . . |] delimiters indicate a mutable array. The secondary Combination constructor creates an initial Combination object and is defined as:

```
new(n : int, k : int) =
  do if n < 0 || k < 0 then failwith
    "Negative argument in Combination ctor"
  do if n < k then failwith
    "Subset size k is larger than n in Combination"
  let starters = [| for i in 0..k-1 -> i |]
  new Combination(n,k,starters)
```

In F#, secondary constructors must call the primary constructor. So I define an int array named starters with values 0 through k-1 and pass it along with n and k to the primary constructor. This F# mechanism is why it is advisable to define any primary constructor as the constructor with the most parameters.

The IsLast member function is defined as:

```
member this.IsLast() : bool =
  if data.[0] = n-k then true
  else false
```

In **Figure 1**, notice that only the last element in a list of all combinations has value n-k located in the first position of the array. F# does not use an explicit return keyword as most languages do; the implied return is the last value in a function, in this case either true or false. The = token checks for equality in F# and is not an assignment operator. The Combination.Successor function is:

```
member this.Successor() : Combination =
  // copy input to temp array
  let temp = [| for i in 0..k-1 -> data.[i] |]
  // find "x" - right-most index to change
  let mutable x = k-1
  while x > 0 && temp.[x] = n - k + x do
    x <- x - 1
  temp.[x] <- temp.[x] + 1 // increment value at x
  // increment all values to the right of x
  for j = x to k-2 do
    temp.[j+1] <- temp.[j] + 1
  // use primary ctor
  let result = new Combination(n, k, temp)
  result
```

I begin by copying the values of the current Combination object context into a mutable array named temp. Next, I define an index variable named x and position it at the end of the temp array. I must use the mutable keyword so that I can decrement this index variable because, by default, most variables in F# are immutable. I use the <- assignment operator.

Once I locate the key index of the current Combination object, I increment that value and all values to the right of the key index. Then I pass the temp array, which now has the value of the successor Combination element, into the primary constructor and return the newly created object.

Notice that I do not return null when I am at the last Combination element—in F# it is considered poor style to do so. The code I present in this article uses a style that is not very F#-ish. F# experts

Figure 3 **The Choose Function**

```
static member Choose(n : int, k : int) : BigInteger =
  if n < 0 || k < 0 then failwith
    "Negative argument in Choose()"
  if n < k then failwith
    "Subset size k is larger than n in Choose()"
  let (delta, iMax) =
    if k < n-k then
      (n-k, k)
    else
      (k, n-k)
  let mutable answer : BigInteger =
    bigint delta + bigint 1
  for i = 2 to iMax do
    answer <- (answer * (bigint delta + bigint i ))
      / bigint i
  answer
```

would likely use a recursive approach, but because I am assuming you are new to F#, I wanted to make my F# code as familiar as possible.

An alternative approach to writing a Successor function is to implement the .NET IEnumerable interface.

The ApplyTo function provides a way to map a combination element to a set of string values:

```
member this.ApplyTo(a : string[]) : string[] =
  if a.Length <> n then failwith
    "Invalid array size in ApplyTo()"
  // array of int
  let result = Array.zeroCreate k
  for i = 0 to k-1 do
    // bind to array of string
    result.[i] <- a.[data.[i]]
  result
```

When performing input argument checks in a member function, I do not need to use the do keyword as is required in type constructors. The static Array.zeroCreate method creates an integer array initialized to all 0 values as you might expect. The ApplyTo function is easy because the range of values in a mathematical combination with subset size k (0..k-1) is exactly the same as the indexes of any .NET array of size k.

The overridden ToString member function simply builds a string made up of the context object's values:

```
override this.ToString() : string =
  let mutable s : string = "^ "
  for i in 0..k-1 do
    s <- s + data.[i].ToString() + " "
  s <- s + "^"
  s
```

I decided to delimit my Combination elements with the ^ (caret) character, which starts with the letter c, and to delimit my Permutation elements with the % (percent) character, which starts with p, to help me identify whether a string of digits represents a Combination or a Permutation object.

The static Choose function is coded as shown in **Figure 3**.

Instead of computing Choose from the definition described earlier in this article, I use two optimizations. First, I use the fact that Choose(n, k) = Choose(n, n-k). For example Choose(9,6) = Choose(9,3). Second, rather than compute three separate factorials, each of which can be very large, I compute a series of partial products. In order to explicitly convert int values to type BigInteger, I use the built-in F# bigint function.

The implementation of the Permutation type is quite similar to the implementation of the Combination type. You can get the complete source code for the CombinationLib library from the Microsoft Code Gallery Web site at code.msdn.microsoft.com.

Figure 4 **Using the CobinatoricsLib**

```
open System
open Module1 // the Combinatorics Lib

try

  printfn "\nBegin combinations and permutations with F# demo\n"
  printfn "All combinations of 5 items 3 at a time in lexicographical
order are: \n"
  let mutable c = new Combination(5,3)
  printfn "%A" c // print initial combination

  // objects cannot be null in F# so use an explicit method
  while c.IsLast() = false do
    c <- c.Successor()
    printfn "%A" c

  printf "\nThe last combination applied to array [| \"ant\"; \"bat\";
\"cow\"; \"dog\"; \"elk\" |] is: \n"
  let animals = [| "ant"; "bat"; "cow"; "dog"; "elk" |]
  //let result =  c.ApplyTo(animals)
  let result = animals |> c.ApplyTo
  printfn "%A" result

  printfn "\nThe number of ways to Choose 200 items 10 at a time =
Choose(200,10) ="
  let Choose_200_10 = Combination.Choose(200,10).ToString("000,000")
  printfn "%s" Choose_200_10

  printfn "\nThe number of ways to arrange 52 cards = 52! = "
  let Factorial_52 = Combination.Factorial(52).ToString("000,000")
  printfn "%s" Factorial_52

  printfn "\nAll permutations of 3 items in lexicographical order are:
\n"
  let mutable p = new Permutation(3)
  printfn "%A" p // print initial permutation
  while p.IsLast() = false do
    p <- p.Successor()
    printfn "%A" p

  printfn "\nEnd demo\n"
  Console.ReadLine() |> ignore

with
  | Failure(errorMsg) -> printfn "Fatal error: %s" errorMsg

// end program
```

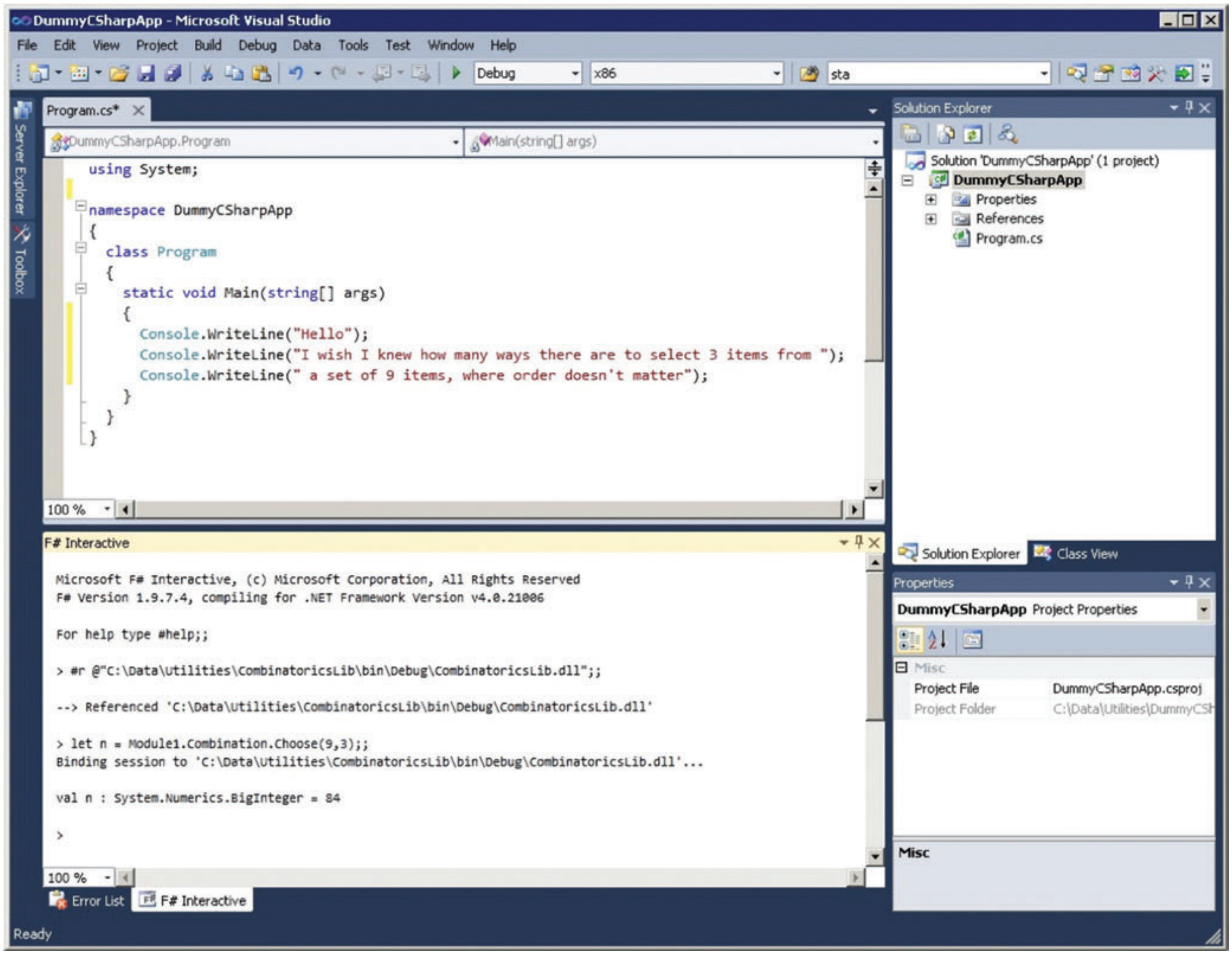
Using the CombinatoricsLib

In this section I explain how to call the function in the CombinatoricsLib library to produce the run shown in the screenshot in **Figure 1**. I begin by launching Visual Studio 2010 and creating a new F# Application project named CombinatoricsDemo. The entire program is listed in **Figure 4**.

Before writing any code, I right-clicked on the Project name in the Solution Explorer window of Visual Studio and selected the Add Reference option from the context menu. I then selected the Browse tab and navigated to the CombinatoricsLib.dll assembly.

I begin the demo program code by adding open statements to the System and Module1 assemblies. Recall that the module name of the CombinatorcsLib is Module1. I wrap all program statements in a try/with block to capture and handle exceptions. I instantiate a Combination object using the secondary constructor to make an initial mathematical combination object c of five items taken three at a time: {0,1,2}. I use the neat F# %A format specifier, which instructs F# to infer how to print my Combination object. I could also have used the %s string format.

Next, I use the F# while..do loop to iterate through and display all 10 Combination(5,3) elements. At this point, the Combination



```
using System;

namespace DummyCSharpApp
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello");
      Console.WriteLine("I wish I knew how many ways there are to select 3 items from ");
      Console.WriteLine(" a set of 9 items, where order doesn't matter");
    }
  }
}
```

```
Microsoft F# Interactive, (c) Microsoft Corporation, All Rights Reserved
F# Version 1.9.7.4, compiling for .NET Framework Version v4.0.21006

For help type #help;;

> #r @"C:\Data\Utilities\CombinatoricsLib\bin\Debug\CombinatoricsLib.dll";;

--> Referenced 'C:\Data\Utilities\CombinatoricsLib\bin\Debug\CombinatoricsLib.dll'

> let n = Module1.Combination.Choose(9,3);;
Binding session to 'C:\Data\Utilities\CombinatoricsLib\bin\Debug\CombinatoricsLib.dll'...

val n : System.Numerics.BigInteger = 84

>
```

Figure 5 **Interactive Use of an F# Library**

object c is the last element and I call the ApplyTo function to map that combination onto an array of strings.

Notice that I call the Choose and Factorial functions from the context of the Combination type rather than the c Combination object. After calling the Permutation type code in a similar way, the demo program concludes by pausing for user input with the Console.ReadLine method, where I pipe the return value to the built-in ignore object. I handle any exceptions in the with block, by simply displaying the exception error message.

In addition to calling an F# library from an F# program as I've just demonstrated, you can call an F# library from any .NET-compliant language. Additionally, Visual Studio allows you to use the handy F# interactive window to make ad hoc calls, as shown in **Figure 5**. In the F# interactive window at the bottom of the screen, I add a reference to the CombinatoricsLib assembly by typing:

```
#r @"C:\(path)\CombinatoricsLib.dll";;
```

In this case #r means add a reference, and ;; terminates an interactive F# statement. Now I can interactively call the functions in the library. Neat!

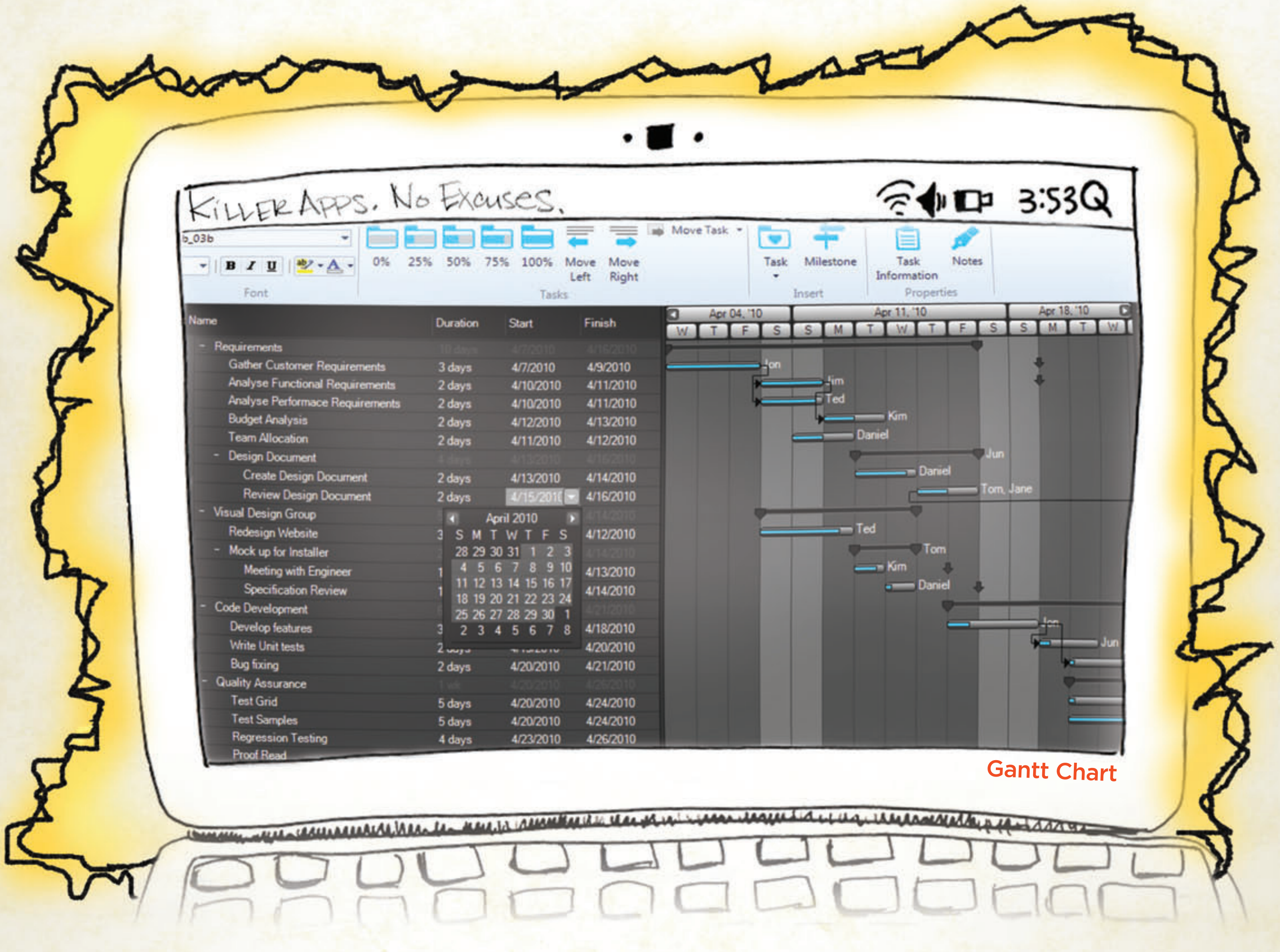
In my opinion, there are several pros and cons to using F#. On the negative side, I found that the learning curve for F# was much steeper than I expected. Writing in a functional style was a big paradigm shift for me. Also, much more so than other languages, F# has multiple ways of coding a particular task, which led me to feel uneasy about whether any F# code I wrote was written in the optimal way.

However, in my case at least, I feel the benefits of learning F# definitely outweigh the costs. When talking to experienced F# coders, most told me that in many cases, even though there are in fact several ways to code a task, the approach taken is more a matter of personal preference than technical efficiency. Also, grappling with F# syntax and coding paradigms (such as default immutability) gave me what I felt was some good insight into coding in procedural languages such as C#. ■

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 has worked on several Microsoft products including Internet Explorer and MSN Search. Dr. McCarthy is the author of ".NET Test Automation Recipes" (Apress, 2006). He can be reached at jammc@microsoft.com.

THANKS to the following technical experts for reviewing this article: *Brian McNamara, Paul Newson and Matthew Podwysocki*

INNOVATION TO SPARK KILLER APPS



Name	Duration	Start	Finish
− Requirements	10 days	4/7/2010	4/16/2010
Gather Customer Requirements	3 days	4/7/2010	4/9/2010
Analyse Functional Requirements	2 days	4/10/2010	4/11/2010
Analyse Performace Requirements	2 days	4/10/2010	4/11/2010
Budget Analysis	2 days	4/12/2010	4/13/2010
Team Allocation	2 days	4/11/2010	4/12/2010
− Design Document	4 days	4/13/2010	4/16/2010
Create Design Document	2 days	4/13/2010	4/14/2010
Review Design Document	2 days	4/15/2010	4/16/2010
− Visual Design Group	[illegible]	[illegible]	4/14/2010
Redesign Website	3	[illegible]	4/12/2010
− Mock up for Installer	[illegible]	[illegible]	4/14/2010
Meeting with Engineer	1	[illegible]	4/13/2010
Specification Review	1	[illegible]	4/14/2010
− Code Development	[illegible]	[illegible]	4/21/2010
Develop features	3	[illegible]	4/18/2010
Write Unit tests	2 days	[illegible]	4/20/2010
Bug fixing	2 days	4/20/2010	4/21/2010
− Quality Assurance	1 wk	4/20/2010	4/26/2010
Test Grid	5 days	4/20/2010	4/24/2010
Test Samples	5 days	4/20/2010	4/24/2010
Regression Testing	4 days	4/23/2010	4/26/2010
Proof Read			

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

Service Bus Buffers

In my October 2009 column, "Routers in the Service Bus" (msdn.microsoft.com/magazine/ee335696), I presented the likely future direction of the Windows Azure AppFabric Service Bus—becoming the ultimate interceptor. I presented the routers feature and promised to write about queues next.

Since then, both routers and queues have been postponed to the second release of the service bus, and instead—for now—the service bus will provide buffers. Future releases will likely add logging, diagnostic and various instrumentation options. I will visit those aspects in a future article. In this article, I'll describe the buffers aspect, and also show you some advanced Windows Communication Foundation (WCF) programming techniques.

Service Bus Buffers

In the service bus, every URI in the service namespace is actually an addressable messaging junction. The client can send a message to that junction, and the junction can relay it to the services. However, each junction can also function as a buffer (see **Figure 1**).

The messages are stored in the buffer for a configurable period of time, even when no service is monitoring the buffer. Note that multiple services can monitor the buffer, but unless you explicitly peek and lock the message, only one of them will be able to retrieve a message.

The client is decoupled from the services behind the buffer, and the client and service need not be running at the same time. Because the client interacts with a buffer and not with an actual service endpoint, all the messages are one-way, and there is no way (out of the box) to obtain the results of the message invocation or any errors.

The service bus buffers should not be equated with queues, such as Microsoft Message Queuing (MSMQ) queues or WCF queued services; they have a number of crucial differences:

- The service bus buffers are not durable, and the messages are stored in memory. This implies a risk of losing messages in the (somewhat unlikely) event of a catastrophic failure of the service bus itself.
- The service bus buffers are not transactional; neither sending nor retrieving messages can be done as part of a transaction.
- The buffers can't handle long-lasting messages. The service must retrieve a message from the buffer within 10 minutes or the message is discarded. Although the WCF MSMQ-based messages also feature a time-to-live, that period is much longer, defaulting to one day. This enables a far broader range of truly disjointed operations and disconnected applications.
- The buffers are limited in size and can't hold more than 50 messages.

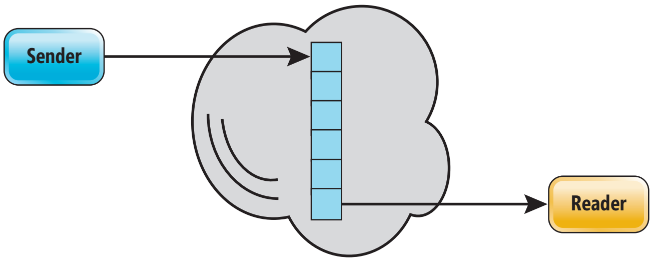


Figure 1 Buffers in the Service Bus

- The buffered messages are capped in size, at 64KB each. Although MSMQ also imposes its own maximum message size, it's substantially larger (4MB per message).

Thus buffers do not provide true queued calls over the cloud; rather, they provide for elasticity in the connection, with calls falling somewhere in between queued calls and fire-and-forget asynchronous calls.

There are two scenarios where buffers are useful. One is an application where the client and the service are interacting over a shaky connection, and dropping the connection and picking it up again is tolerated as long as the messages are buffered during the short offline period. A second (and more common) scenario is a client issuing asynchronous one-way calls and utilizing a response buffer (as described later in the Response Service section) to handle the results of the calls. Such interaction is like viewing the network connection more as a bungee cord rather than a rigid network wire that has no storage capacity.

Working with Buffers

The buffer address must be unique; you can have only a single buffer associated with an address and the address can't already be used by a buffer or a service. However, multiple parties can retrieve messages from the same buffer. In addition, the buffer address must use either HTTP or HTTPS for the scheme. To send and retrieve messages from the buffer, the service bus offers an API similar to that of System.Messaging; that is, it requires you to interact with raw messages. The service bus administrator manages the buffers independently of services or clients. Each buffer must have a policy governing its behavior and lifetime. Out of the box, the service bus administrator must perform programmatic calls to create and manage buffers.

Each buffer policy is expressed via an instance of the MessageBufferPolicy class as shown in **Figure 2**.

Code download available at code.msdn.microsoft.com/mag201005ServBus.

The Discoverability policy property is an enum of the type DiscoverabilityPolicy, controlling whether or not the buffer is included in the service bus registry (the ATOM feed):

```
public enum DiscoverabilityPolicy
{
  Managers,
  ManagersListeners,
  ManagersListenersSenders,
  Public
}
```

Discoverability defaults to DiscoverabilityPolicy.Managers, which means it requires a managed authorization claim. Setting it to DiscoverabilityPolicy.Public publishes it to the feed without any authorization.

The ExpiresAfter property controls the lifetime of messages in the buffer. The default is five minutes, the minimum value is one minute and the maximum allowed value is 10 minutes. Any attempt to configure a longer lifetime is silently ignored.

The MaxMessageCount property caps the buffer size. The policy defaults to 10 messages, and the minimum value is, of course, set to one. As mentioned already, the maximum buffer size is 50, and attempts to configure a larger size are silently ignored.

The OverflowPolicy property is an enum with a single value defined as:

```
public enum OverflowPolicy
{
  RejectIncomingMessage
}
```

OverflowPolicy controls what to do with the message when the buffer is maxed out; that is, when it's already filled to capacity (defined by MaxMessageCount). The only possible option is to reject the message—send it back with an error to the sender.

The single-value enum serves as a placeholder for future options, such as discarding the message without informing the sender or removing messages from the buffer and accepting the new message.

The last two properties are responsible for security configuration. The AuthorizationPolicy property instructs the service bus whether or not to authorize the client's token:

```
public enum AuthorizationPolicy
{
  NotRequired,
  RequiredToSend,
  RequiredToReceive,
  Required
}
```

The default value of AuthorizationPolicy.Required requires authorizing both sending and receiving clients.

Finally, the TransportProtection property stipulates the minimum level of transfer security for the message to the buffer, using an enum of the type TransportProtectionPolicy:

```
public enum TransportProtectionPolicy
{
  None,
  AllPaths,
}
```

Transport security via TransportProtectionPolicy.AllPaths is the default for all buffer policies, and it mandates the use of an HTTPS address.

You can use the MessageBufferClient class to administer your buffer, as shown in **Figure 3**.

You use the static methods of MessageBufferClient to obtain an authenticated instance of MessageBufferClient by providing the static methods with the service bus credentials (of the type TransportClientEndpointBehavior). Whenever using MessageBufferClient, you typically need to check if the buffer already exists in the service bus by calling the GetMessageBuffer method. If there's no buffer, GetMessageBuffer throws an exception.

Figure 2 **The MessageBufferPolicy Class**

```
[DataContract]
public class MessageBufferPolicy : ...
{
  public MessageBufferPolicy();
  public MessageBufferPolicy(MessageBufferPolicy policyToCopy);

  public DiscoverabilityPolicy Discoverability
  {get;set;}

  public TimeSpan ExpiresAfter
  {get;set;}

  public int MaxMessageCount
  {get;set;}

  public OverflowPolicy OverflowPolicy
  {get;set;}

  public AuthorizationPolicy Authorization
  {get;set;}

  public TransportProtectionPolicy TransportProtection
  {get;set;}
}
```

Here's how to create a buffer programmatically:

```
Uri bufferAddress =
  new Uri(@"https://MyNamespace.servicebus.windows.net/MyBuffer/");

TransportClientEndpointBehavior credential = ...

MessageBufferPolicy bufferPolicy = new MessageBufferPolicy();

bufferPolicy.MaxMessageCount = 12;
bufferPolicy.ExpiresAfter = TimeSpan.FromMinutes(3);
bufferPolicy.Discoverability = DiscoverabilityPolicy.Public;

MessageBufferClient.CreateMessageBuffer(credential,bufferAddress,
  bufferPolicy);
```

In this example, you instantiate a buffer policy object and set the policy to some desired values. All it takes to install the buffer is calling the CreateMessageBuffer method of MessageBufferClient with the policy and some valid credentials.

As an alternative to programmatic calls, you can use my Service Bus Explorer (presented in my routers article and also available online with the sample code for this article) to both view and modify buffers. **Figure 4** shows how to create a new buffer by specifying its address and various policy properties. In much the same way, you can also delete all buffers in the service namespace.

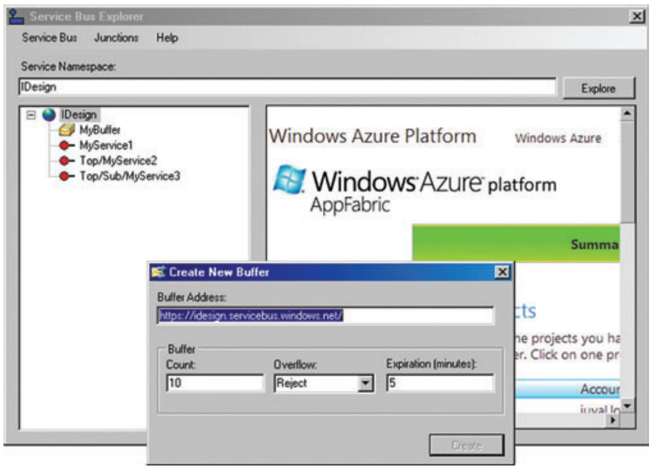
Figure 3 **The MessageBufferClient Class**

```
public sealed class MessageBufferClient
{
  public Uri MessageBufferUri
  {get;}

  public static MessageBufferClient CreateMessageBuffer(
    TransportClientEndpointBehavior credential,
    Uri messageBufferUri,MessageBufferPolicy policy);

  public static MessageBufferClient GetMessageBuffer(
    TransportClientEndpointBehavior credential,Uri messageBufferUri);
  public MessageBufferPolicy GetPolicy();
  public void DeleteMessageBuffer();

  // More members
}
```



Service Namespace: IDesign

Buffer Address: https://idesign.servicebus.windows.net/

Count: 10
Overflow: Reject
Expiration (minutes): 5

Figure 4 **Creating a Buffer Using the Service Bus Explorer**

You can also review and modify the policies of existing buffers, purge messages from the buffer and even delete a buffer by selecting the buffer in the service namespace tree and interacting with the buffer properties in the right pane, as shown in **Figure 5**.

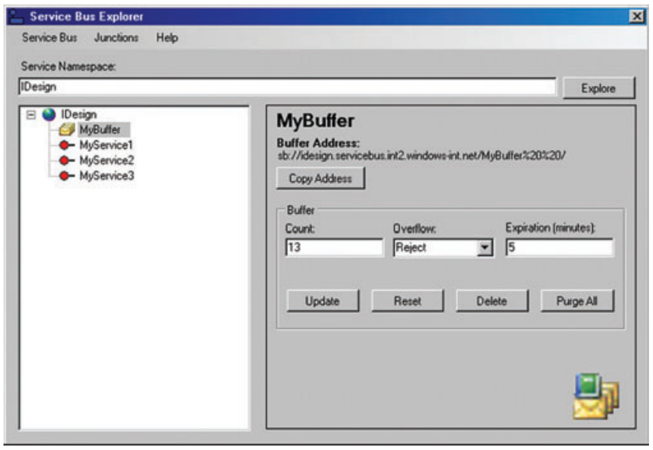
Streamlining Administration

When creating buffers, it's best to maximize both the buffer size and its lifespan, to give the clients and services more time to interact. Moreover, it's a good idea to make the buffer discoverable so you can view it on the service bus registry. When it comes to using the buffer, both the client and the service should verify that the buffer is already created, or else proceed to create it.

To automate these steps, I created the ServiceBusHelper class:

```
public static partial class ServiceBusHelper
{
  public static void CreateBuffer(string bufferAddress,string secret);
  public static void CreateBuffer(string bufferAddress,string issuer,
    string secret);

  public static void VerifyBuffer(string bufferAddress,string secret);
  public static void VerifyBuffer(string bufferAddress,string issuer,
    string secret);
  public static void PurgeBuffer(Uri bufferAddress,
    TransportClientEndpointBehavior credential);
  public static void DeleteBuffer(Uri bufferAddress,
    TransportClientEndpointBehavior credential);
}
```



Service Namespace: IDesign

Buffer Address:
sb://idesign.servicebus.int2.windows-int.net/MyBuffer%20%20/

Count: 13
Overflow: Reject
Expiration (minutes): 5

Figure 5 **A Buffer in the Service Bus Explorer**

The CreateBuffer method creates a new discoverable buffer with a maximum capacity of 50 messages and a duration of 10 minutes. If the buffer already exists, CreateBuffer deletes the old buffer. The VerifyBuffer method verifies that a buffer exists and, if it doesn't, creates a new buffer. PurgeBuffer is useful for purging all buffered messages during diagnostics or debugging. DeleteBuffer simply deletes the buffer. **Figure 6** shows partial listing of the implementation of these methods.

The BufferExists method uses the GetPolicy method of MessageBufferClient to see if a buffer exists, and it interprets an error as an indication that the buffer does not exist. Purging a buffer is done by copying its policy, deleting the buffer and creating a new buffer (with the same address) *with* the old policy.

Sending and Retrieving Messages

As mentioned already, the service bus buffers require interactions with raw WCF messages. This is done with the Send and Retrieve methods of MessageBufferClient (obtained when creating or getting a buffer):

```
public sealed class MessageBufferClient
{
  public void Send(Message message);
  public void Send(Message message,TimeSpan timeout);

  public Message Retrieve();
  public Message Retrieve(TimeSpan timeout);

  // More members
}
```

Both methods are subject to a timeout that defaults to one minute for the parameter-less versions. For the sender, the timeout means how long to wait in case the buffer is full. For the retriever, the timeout means how long to wait in case the buffer is empty.

Here's the sender-side code for sending raw messages to the buffer:

```
TransportClientEndpointBehavior credential = ...;
Uri bufferUri = new Uri(@"sb://MyNamespace.servicebus.windows.net/MyBuffer/");

MessageBufferClient client =
  MessageBufferClient.GetMessageBuffer(credential,bufferUri);

Message message = Message.CreateMessage(MessageVersion.Default,"Hello");

client.Send(message,TimeSpan.MaxValue);
```

The sender first creates a credentials object and uses it to obtain an instance of MessageBufferClient. The sender then creates a WCF message and sends it to the buffer. Here is the retrieving-side code for retrieving raw messages from the buffer:

```
TransportClientEndpointBehavior credential = ...;
Uri bufferUri = new Uri(@"sb://MyNamespace.servicebus.windows.net/MyBuffer/");

MessageBufferClient client =
  MessageBufferClient.GetMessageBuffer(credential,bufferUri);
Message message = client.Retrieve();

Debug.Assert(message.Headers.Action == "Hello");
```

Buffered Services

Using raw WCF messages as in the preceding code snippets is what the service bus has to offer. And yet, such a programming model leaves much to be desired. It's cumbersome, tedious, non-structured, not object-oriented and not type safe. It's a throwback to the days before WCF itself, with explicit programming against MSMQ using the System.Messaging API. You need to parse the message content and switch on its elements.

Fortunately, you can improve on the basic offering. Instead of interacting with raw messages, you should elevate the interaction to structured calls between clients and services. Although this requires a considerable degree of low-level advanced work, I was able to encapsulate it with a small set of helper classes.

To provide for structured buffered calls on the service side, I wrote BufferedServiceBusHost<T> defined as:

```
// Generic type parameter based host
public class ServiceHost<T> : ServiceHost
{...}

public class BufferedServiceBusHost<T> : ServiceHost<T>,...
{
  public BufferedServiceBusHost(params Uri[] bufferAddresses);
  public BufferedServiceBusHost(
    T singleton,params Uri[] bufferAddresses);

  /* Additional constructors */
}
```

I modeled BufferedServiceBusHost<T> after using WCF with the MSMQ binding. You need to provide its constructor with the address or addresses of the buffers to retrieve messages from. The rest is just as with a regular WCF service host:

```
Uri buffer = new Uri(@"https://MyNamespace.servicebus.windows.net/MyBuffer");
ServiceHost host = new BufferedServiceBusHost<MyService>(buffer);
host.Open();
```

Note that you can provide the constructors with multiple buffer addresses to monitor, just like a WCF service host can open multiple endpoints with different queues. There's no need (or way) to provide any of these buffer addresses in the service endpoint section in the config file (although the buffer addresses can come from the app settings section if you so design).

While the actual communication with the service bus buffer is done with raw WCF messages, that work is encapsulated. BufferedServiceBusHost<T> will verify that the buffers provided actually exist and will create them if they don't, using the buffer policy of ServiceBusHelper.VerifyBuffer shown in **Figure 6**. BufferedServiceBusHost<T> will use the default transfer security of securing all paths. It will also verify that the contracts of the provided service generic type parameter T are all one-way; that is, they all have only one-way operations (just as the one-way relay binding does). One last feature: when closing the host, in debug builds only, BufferedServiceBusHost<T> will purge all its buffers to ensure a smooth start for the next debug session.

BufferedServiceBusHost<T> operates by hosting the specified service locally. For each service contract on the type parameter T, BufferedServiceBusHost<T> adds an endpoint over IPC (named pipes). The IPC binding to those endpoints is configured to never time out.

Although IPC always has a transport session, to mimic MSMQ behavior even per-session services are treated as per-call services. Each dequeued WCF message is played to a new instance of the service, potentially concurrently with previous messages, just as with the MSMQ binding. If the provided service type is a singleton, BufferedServiceBusHost<T> respects that and will send all messages across all buffers and endpoints to the same service instance, just as with the MSMQ binding.

BufferedServiceBusHost<T> monitors each specified buffer on the separate background worker thread. When a message is deposited in the buffer, BufferedServiceBusHost<T> retrieves it and converts the raw WCF message into a call to the appropriate endpoint over IPC.

Figure 6 **Partial Listing of the Buffer Helper Methods**

```
public static partial class ServiceBusHelper
{
  public static void CreateBuffer(string bufferAddress,
    string issuer,string secret)
  {
    TransportClientEndpointBehavior credentials = ...;
    CreateBuffer(bufferAddress,credentials);
  }
  static void CreateBuffer(string bufferAddress,
    TransportClientEndpointBehavior credentials)
  {
    MessageBufferPolicy policy = CreateBufferPolicy();
    CreateBuffer(bufferAddress,policy,credentials);
  }
  static internal MessageBufferPolicy CreateBufferPolicy()
  {
    MessageBufferPolicy policy = new MessageBufferPolicy();
    policy.Discoverability = DiscoverabilityPolicy.Public;
    policy.ExpiresAfter = TimeSpan.Fromminutes(10);
    policy.MaxMessageCount = 50;

    return policy;
  }
   public static void PurgeBuffer(Uri bufferAddress,
     TransportClientEndpointBehavior credentials)
   {
     Debug.Assert(BufferExists(bufferAddress,credentials));
     MessageBufferClient client =
       MessageBufferClient.GetMessageBuffer(credentials,bufferAddress);
     MessageBufferPolicy policy = client.GetPolicy();
     client.DeleteMessageBuffer();

    MessageBufferClient.CreateMessageBuffer(credential,bufferAddress,policy);
   }
   public static void VerifyBuffer(string bufferAddress,
     string issuer,string secret)
   {
     TransportClientEndpointBehavior credentials = ...;
     VerifyBuffer(bufferAddress,credentials);
   }
  internal static void VerifyBuffer(string bufferAddress,
    TransportClientEndpointBehavior credentials)
  {
    if(BufferExists(bufferAddress,credentials))
    {
      return;
    }
    CreateBuffer(bufferAddress,credentials);
  }
  internal static bool BufferExists(Uri bufferAddress,
    TransportClientEndpointBehavior credentials)
  {
    try
    {
      MessageBufferClient client =
        MessageBufferClient.GetMessageBuffer(credentials,bufferAddress);
      client.GetPolicy();
      return true;
    }
    catch(FaultException)
    {}

    return false;
  }
  static void CreateBuffer(string bufferAddress,
    MessageBufferPolicy policy,
    TransportClientEndpointBehavior credentials)
  {
    Uri address = new Uri(bufferAddress);
    if(BufferExists(address,credentials))
    {
      MessageBufferClient client =
        MessageBufferClient.GetMessageBuffer(credentials,address);
      client.DeleteMessageBuffer();
    }
    MessageBufferClient.CreateMessageBuffer(credentials,address,policy);
  }
}
```

Figure 7 **Partial Listing of BufferedServiceBusHost<T>**

```
public class BufferedServiceBusHost<T> :
  ServiceHost<T>,IServiceBusProperties
{
  Uri[] m_BufferAddresses;
  List<Thread> m_RetrievingThreads;
  IChannelFactory<IDuplexSessionChannel>
    m_Factory;
  Dictionary<string,IDuplexSessionChannel>
    m_Proxies;

  const string CloseAction =
    "BufferedServiceBusHost.CloseThread";

  public BufferedServiceBusHost(params Uri[]
    bufferAddresses)
  {
    m_BufferAddresses = bufferAddresses;
    Binding binding = new NetNamedPipeBinding();
    binding.SendTimeout = TimeSpan.MaxValue;

    Type[] interfaces =
      typeof(T).GetInterfaces();

    foreach(Type interfaceType in interfaces)
    {
      VerifyOneway(interfaceType);
      string address =
        @"net.pipe://localhost/" + Guid.NewGuid();
      AddServiceEndpoint(interfaceType,binding,
        address);
    }
    m_Factory =
      binding.BuildChannelFactory
      <IDuplexSessionChannel>();
    m_Factory.Open();
  }
  protected override void OnOpened()
  {
    CreateProxies();
    CreateListeners();
    base.OnOpened();
  }
  protected override void OnClosing()
  {
    CloseListeners();

    foreach(IDuplexSessionChannel proxy in
      m_Proxies.Values)
    {
      proxy.Close();
    }

    m_Factory.Close();

    PurgeBuffers();

    base.OnClosing();
  }

  // Verify all operations are one-way

  void VerifyOneway(Type interfaceType)
  {...}
  void CreateProxies()
  {
    m_Proxies =
      new Dictionary
      <string,IDuplexSessionChannel>();

    foreach(ServiceEndpoint endpoint in
      Description.Endpoints)
    {
      IDuplexSessionChannel channel =
        m_Factory.CreateChannel(endpoint.Address);
      channel.Open();
      m_Proxies[endpoint.Contract.Name] =
        channel;
    }
  }

  void CreateListeners()
  {
    m_RetrievingThreads = new List<Thread>();

    foreach(Uri bufferAddress in
      m_BufferAddresses)
    {
      ServiceBusHelper.VerifyBuffer(
        bufferAddress.AbsoluteUri,m_Credential);

      Thread thread = new Thread(Dequeue);

      m_RetrievingThreads.Add(thread);
      thread.IsBackground = true;
      thread.Start(bufferAddress);
    }
  }

  void Dequeue(object arg)
  {
    Uri bufferAddress = arg as Uri;

    MessageBufferClient bufferClient =
      MessageBufferClient.GetMessageBuffer(
        m_Credential,bufferAddress);
    while(true)
    {
      Message message =
        bufferClient.Retrieve(TimeSpan.MaxValue);
      if(message.Headers.Action == CloseAction)
      {
        return;
      }
      else
      {
        Dispatch(message);
      }
    }
  }

  void Dispatch(Message message)
  {
    string contract = ExtractContract(message);
    m_Proxies[contract].Send(message);
  }
  string ExtractContract(Message message)
  {
    string[] elements =
      message.Headers.Action.Split('/');
    return elements[elements.Length-2];
  }
  protected override void OnClosing()
  {
    CloseListeners();
    foreach(IDuplexSessionChannel proxy in
      m_Proxies.Values)
    {
      proxy.Close();
    }
    m_Factory.Close();

    PurgeBuffers();
    base.OnClosing();
  }
  void SendCloseMessages()
  {
    foreach(Uri bufferAddress in
      m_BufferAddresses)
    {
      MessageBufferClient bufferClient =
        MessageBufferClient.GetMessageBuffer(
        m_Credential,bufferAddress);
      Message message =
        Message.CreateMessage(
        MessageVersion.Default,CloseAction);
      bufferClient.Send(message);
    }
  }
  void CloseListeners()
  {
    SendCloseMessages();

    foreach(Thread thread in m_RetrievingThreads)
    {
      thread.Join();
    }
  }

  [Conditional("DEBUG")]
  void PurgeBuffers()
  {
    foreach(Uri bufferAddress in
      m_BufferAddresses)
    {
      ServiceBusHelper.PurgeBuffer(
        bufferAddress,m_Credential);
    }
  }
}
```

Figure 7 provides a partial listing of BufferedServiceBusHost<T>, with most of the error handling and security removed.

BufferedServiceBusHost<T> stores the proxies to the locally hosted IPC endpoints in a dictionary called m_Proxies:

```
Dictionary<string,IDuplexSessionChannel> m_Proxies;
```

The key into the dictionary is the endpoints' contract type name.

The constructors store the provided buffer addresses and then use reflection to obtain a collection of all the interfaces on the service type. For each interface, BufferedServiceBusHost<T> verifies it has only one-way operations, then calls the base AddService-Endpoint to add an endpoint for that contract type. The address is an IPC address using a GUID for the pipe's name. The constructors use the IPC binding to build a channel factory of the type IChannel-Factory<IDuplexSessionChannel>. IChannelFactory<T> is used to create a non-strongly typed channel over the binding:

```
public interface IChannelFactory<T> : IChannelFactory
{
  T CreateChannel(EndpointAddress to);
  // More members
}
```

After opening the internal host with all its IPC endpoints, the OnOpened method creates the internal proxies to those endpoints and the buffered listeners. These two steps are the heart of BufferedServiceBusHost<T>. To create the proxies, it iterates over the collection of endpoints. It obtains each endpoint's address and

Figure 8 **Partial Listing of BufferedServiceBusClient<T>**

```
public abstract class BufferedServiceBusClient<T> :
  HeaderClientBase<T,ResponseContext>,IServiceBusProperties where T : class
{
  MessageBufferClient m_BufferClient;

  public BufferedServiceBusClient(Uri bufferAddress) :
    base(new NetOnewayRelayBinding(),new EndpointAddress(bufferAddress))
  {}

  protected virtual void Enqueue(Action action)
  {
    try
    {
      action();
    }
    catch(InvalidOperationException exception)
    {
      Debug.Assert(exception.Message ==
        "This message cannot support the operation " +
        "because it has been written.");
    }
  }
  protected override T CreateChannel()
  {
   ServiceBusHelper.VerifyBuffer(Endpoint.Address.Uri.AbsoluteUri,Credential);
    m_BufferClient =
      MessageBufferClient.GetMessageBuffer(Credential,m_BufferAddress);

    return base.CreateChannel();
  }
  protected override void PreInvoke(ref Message request)
  {
    base.PreInvoke(ref request);

    m_BufferClient.Send(request);
  }
  protected TransportClientEndpointBehavior Credential
  {
    get
    {...}
    set
    {...}
  }
}
```

uses the IChannelFactory<IDuplexSessionChannel> to create a channel against that address. That channel (or proxy) is then stored in the dictionary. The CreateListeners method iterates over the specified buffer addresses. For each address, it verifies the buffer and creates a worker thread to dequeue its messages.

The Dequeue method uses a MessageBufferClient to retrieve the messages in an infinite loop and dispatch them using the Dispatch method. Dispatch extracts from the message the target contract name and uses it to look up the IDuplexChannel from the proxies dictionary and send the message over IPC. IDuplexChannel is supported by the underlying IPC channel and it provides for a way to send raw messages:

```
public interface IOutputChannel : ...
{
  void Send(Message message,TimeSpan timeout);
  // More members
}
public interface IDuplexSessionChannel : IOutputChannel,...
{}
```

If an error occurred during the IPC call, BufferedServiceBusHost<T> will recreate the channel it manages against that endpoint (not shown in **Figure 7**). When you close the host, you need to close the proxies. This will gracefully wait for the calls in progress to complete. The problem is how to gracefully close all the retrieving threads, because MessageBufferClient.Retrieve is a blocking operation and there is no built-in way to abort it. The solution is to post to each monitored buffer a special private message whose action signals the retrieving thread to exit. This is what the SendCloseMessages method does. The CloseListeners method posts that private message to the buffers and then waits for all the listening threads to terminate by joining them. Closing the listening threads stops feeding messages to the internal proxies, and once the proxies are closed (when all current calls in progress have returned), the host is ready to shut down. BufferedServiceBusHost<T> also supports an ungraceful Abort method that just aborts all threads (not shown in **Figure 7**).

Finally, note that BufferedServiceBusHost<T> supports the interface IServiceBusProperties I defined as:

```
public interface IServiceBusProperties
{
  TransportClientEndpointBehavior Credential
  {get;set;}

  Uri[] Addresses
  {get;}
}
```

I needed such an interface in a few places in building my framework, especially in streamlining buffering. For the client, I wrote the class BufferedServiceBusClient<T> defined as:

```
public abstract class BufferedServiceBusClient<T> :
  HeaderClientBase<T,ResponseContext>,IServiceBusProperties
{
  // Buffer address from config
  public BufferedServiceBusClient()
  {}
  // No need for config file
  public BufferedServiceBusClient(Uri bufferAddress);


  /* Additional constructors with different credentials */
  protected virtual void Enqueue(Action action);
}
```

BufferedServiceBusClient<T> derives from my HeaderClientBase<T,H> (a helper proxy used to pass information in the message headers; see my November 2007 article, "Synchronization Contexts in WCF," available at msdn.microsoft.com/magazine/cc163321):

```
public abstract class HeaderClientBase<T,H> : InterceptorClientBase<T>
                                              where T : class
{
  protected H Header
  {get;set;}

  // More members
}
```

The purpose of that base class is to support a response service, as discussed in the following section. For a plain client of a buffered service, that derivation is immaterial.

You can use BufferedServiceBusClient<T> with or without a client config file. The constructors that accept the buffer address

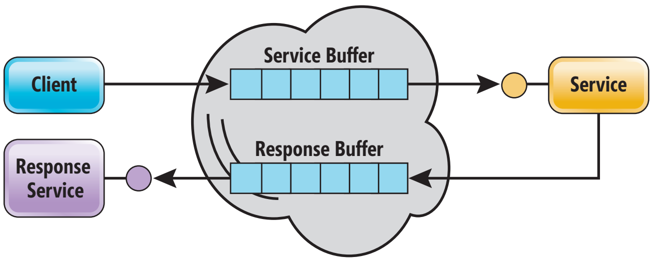


Figure 9 Service Bus Buffered Response Service

Figure 10 **Streamlining the Client Side**

```
[ServiceContract]
interface ICalculator
{
  [OperationContract(IsOneWay = true)]
  void Add(int number1,int number2);
}

class CalculatorClient : ClientBufferResponseBase<ICalculator>,ICalculator
{
  public CalculatorClient(Uri responseAddress) : base(responseAddress)
  {}

  public void Add(int number1,int number2)
  {
     Enqueue(()=>Channel.Add(number1,number2));
  }
}
```

do not require a config file. The parameter-less constructor or the constructors that accept the endpoint name expect the config file to contain an endpoint matching the contract type with the one-way relay binding (although that binding is completely ignored by BufferedServiceBusClient<T>).

When deriving your proxy from BufferedServiceBusClient<T>, you will need to use the protected Enqueue method instead of directly using the Channel property:

```
[ServiceContract]
interface IMyContract
{
  [OperationContract(IsOneWay = true)]
  void MyMethod(int number);
}

class MyContractClient : BufferedServiceBusClient<IMyContract>,IMyContract
{
  public void MyMethod(int number)
  {
    Enqueue(()=>Channel.MyMethod(number));
  }
}
```

Enqueue accepts a delegate (or a lambda expression) that wraps the use of the Channel property. The result is still type safe. **Figure 8** shows a partial listing of the BufferedServiceBusClient<T> class.

The constructors of BufferedServiceBusClient<T> supply its base constructor with the buffer address and the binding, which is always a one-way relay binding to enforce the one-way operations validation. The CreateChannel method verifies that the target buffer exists and obtains a MessageBufferClient representing it. The heart of BufferedServiceBusClient<T> is the PreInvoke method. PreInvoke is a virtual method provided by InterceptorClientBase<T>, the base class of HeaderClientBase<T,H>:

```
public abstract class InterceptorClientBase<T> : ClientBase<T> where T : class
{
  protected virtual void PreInvoke(ref Message request);
  // Rest of the implementation
}
```

PreInvoke allows you to easily process the WCF messages before they're dispatched by the client. BufferedServiceBusClient<T> overrides PreInvoke and uses the buffer client to send the message to the buffer. That way, the client maintains a structured programming model and BufferedServiceBusClient<T> encapsulates the interaction with the WCF message. The downside is that the message can only be sent once, and when the root class of ClientBase tries to send it, it throws an InvalidOperationException. This is where Enqueue comes in handy by snuffing out that exception.

Response Service

In my February 2007 column, "Build a Queued WCF Response Service" (msdn.microsoft.com/magazine/cc163482), I explained that the only way to receive the result (or errors) of a queued call is to use a queued response service. I showed how to pass in the message headers a response context object that contains the logical method ID and the response address:

```
[DataContract]
public class ResponseContext
{
  [DataMember]
  public readonly string ResponseAddress;

  [DataMember]
  public readonly string MethodId;

  public ResponseContext(string responseAddress,string methodId);

  public static ResponseContext Current
  {get;set;}

  // More members
}
```

It's best to maximize the buffer size and its lifespan to give clients and services more time to interact.

The same design pattern holds true when dealing with buffers. The client needs to provide a dedicated response buffer for the service to buffer the response to. The client also needs to pass the response address and the method ID in the message headers, just as with the MSMQ-based calls. The main difference between MSMQ-based response service and the service bus is that the response buffer must also reside

Figure 11 **Implementing ClientBufferResponseBase<T>**

```
public abstract class ClientBufferResponseBase<T> :
  BufferedServiceBusClient<T> where T : class
{
  public readonly Uri ResponseAddress;

  public ClientBufferResponseBase(Uri responseAddress)
  {
    ResponseAddress = responseAddress;
  }

  /* More Constructors */

  protected override void PreInvoke(ref Message request)
  {
    string methodId = GenerateMethodId();
    Header = new ResponseContext(ResponseAddress.AbsoluteUri,methodId);
    base.PreInvoke(ref request);
  }

  protected virtual string GenerateMethodId()
  {
    return Guid.NewGuid().ToString();
  }

  // Rest of the implementation
}
```

Figure 12 **The ServiceBufferResponseBase<T> Class**

```
public abstract class ServiceBufferResponseBase<T> :
  BufferedServiceBusClient<T> where T : class
{
  public ServiceBufferResponseBase() :
   base(new Uri(ResponseContext.Current.ResponseAddress))
 {
   Header = ResponseContext.Current;

   // Grab the credentials the host was using

   IServiceBusProperties properties =
     OperationContext.Current.Host as IServiceBusProperties;
   Credential = properties.Credential;
  }
}
```

in the service bus, as shown in **Figure 9**. To streamline the client side, I wrote the class ClientBufferResponseBase<T> defined as:

```
public abstract class ClientBufferResponseBase<T> :
  BufferedServiceBusClient<T> where T : class
{
  protected readonly Uri ResponseAddress;

  public ClientBufferResponseBase(Uri responseAddress);

  /* Additional constructors with different credentials */

  protected virtual string GenerateMethodId();
}
```

ClientBufferResponseBase<T> is a specialized subclass of BufferedServiceBusClient<T>, and it adds the response context to the message headers. This is why I made BufferedServiceBusClient<T> derive from HeaderClientBase<T,H> and not merely from InterceptorClientBase<T>. You can use ClientBufferResponseBase<T> just like BufferedServiceBusClient, as shown in **Figure 10**. Using the subclass of ClientBufferResponseBase<T> is straightforward:

```
Uri resposeAddress =
  new Uri(@"sb://MyNamespace.servicebus.windows.net/MyResponseBuffer/");

CalculatorClient proxy = new CalculatorClient(responseAddress);
proxy.Add(2,3);
proxy.Close();
```

It's handy when managing the responses on the client side to have the invoking client obtain the method ID used to dispatch the call. This is easily done via the Header property:

```
CalculatorClient proxy = new CalculatorClient(responseAddress);
proxy.Add(2,3);
string methodId = proxy.Header.MethodId;
```

Figure 11 lists the implementation of ClientBufferResponseBase<T>.

ClientBufferResponseBase<T> overrides the PreInvoke method of HeaderClientBase<T,H> so that it could generate a new method ID for each call and set it into the headers. To streamline the work required by the buffered service to call the response service, I wrote the class ServiceBufferResponseBase<T> shown in **Figure 12**.

Although the service could use a plain BufferedServiceBusClient<T> to enqueue the response, you will need to extract the response buffer address from the headers and somehow obtain the credentials to log into the service bus buffer. You will also need to provide the headers of the outgoing call with the response context. All these steps can be streamlined with ServiceBufferResponseBase<T>. ServiceBufferResponseBase<T> provides its base constructor with the address out of the response context, and it also sets that context into the outgoing headers.

Another simplifying assumption ServiceBufferResponseBase<T> makes is that the responding service can use the same credentials its host used (to retrieve messages from its own buffer) to send messages to the response buffer. To that end, ServiceBufferResponseBase<T> obtains a reference to its own host from the operation context and reads the credentials using the IServiceBusProperties implementation of the host. ServiceBufferResponseBase<T> copies those credentials for its own use (done inside BufferedServiceBusClient<T>). This, of course, mandates the use of BufferedServiceBusHost<T> to host the service in the first place. Your service needs to derive a proxy class from ServiceBufferResponseBase<T> and use it to respond. For example, given this response contract:

```
[ServiceContract]
interface ICalculatorResponse
{
  [OperationContract(IsOneWay = true)]
  void OnAddCompleted(int result,ExceptionDetail error);
}
```

This would be the definition of the proxy to the response service:

```
class CalculatorResponseClient :
  ServiceBufferResponseBase<ICalculatorResponse>,ICalculatorResponse
{
  public void OnAddCompleted(int result,ExceptionDetail error)
  {
    Enqueue(()=>Channel.OnAddCompleted(result,error));
  }
}
```

Figure 13 shows a simple buffered service responding to its client.

All the response service needs is to access the method ID from the message headers as shown here:

```
class MyCalculatorResponse : ICalculatorResponse
{
  public void OnAddCompleted(int result,ExceptionDetail error)
  {
    string methodId = ResponseContext.Current.MethodId;
    ...
  }
}
```

Stay tuned for further exploration of the service bus. ■

Figure 13 **Using ServiceBufferResponseBase<T>**

```
class MyCalculator : ICalculator
{
  [OperationBehavior(TransactionScopeRequired = true)]
  public void Add(int number1,int number2)
  {
    int result = 0;
    ExceptionDetail error = null;
    try
    {
      result = number1 + number2;
    }
     // Don't rethrow
    catch(Exception exception)
    {
      error = new ExceptionDetail(exception);
    }
    finally
    {
      CalculatorResponseClient proxy = new CalculatorResponseClient();
      proxy.OnAddCompleted(result,error);
      proxy.Close();
    }
  }
}
```

JUVAL LOWY *is a software architect with IDesign providing WCF training and architecture consulting. This article contains excerpts from his recent book, "Programming WCF Services, Third Edition" (O'Reilly, 2010). He's also the Microsoft regional director for the Silicon Valley. Contact Lowy at idesign.net.*

THANKS to the following technical expert for reviewing this article:
Jeanne Baker

Regular Expression Denial of Service Attacks and Defenses

In the November 2009 issue, I wrote an article titled "XML Denial of Service Attacks and Defenses" (msdn.microsoft.com/magazine/ee335713), in which I described some particularly effective denial of service (DoS) attack techniques against XML parsers. I received a lot of e-mail about this article from readers wanting to know more, which really encourages me that people understand how serious DoS attacks can be.

I believe that in the next four to five years, as privilege escalation attacks become more difficult to execute due to increased adoption of memory protections such as Data Execution Prevention (DEP), Address Space Layout Randomization (ASLR), and isolation and privilege reduction techniques, attackers will shift their focus to DoS blackmail attacks. Developers can continue to protect their applications by staying ahead of the attack trend curve and addressing potential future DoS vectors today.

One of those potential future DoS vectors is the regular expression DoS. At the Open Web Application Security Project (OWASP) Israel Conference 2009, Checkmarx Chief Architect Alex Roichman and Senior Programmer Adar Weidman presented some excellent research on the topic of regular expression DoS, or "ReDoS." Their research revealed that a poorly written regular expression can be exploited so that a relatively short attack string (fewer than 50 characters) can take hours or more to evaluate. In the worst-case scenario, the processing time is actually exponential to the number of characters in the input string, meaning that adding a single character to the string doubles the processing time.

A poorly written regular expression can be exploited so that a relatively short attack string can take hours or more to evaluate.

In this article, I will describe what makes a regex vulnerable to these attacks. I will also present code for a Regex Fuzzer, a test utility designed to identify vulnerable regexes by evaluating them against thousands of random inputs and flagging whether any of the inputs take an unacceptably long time to complete processing.

(Note: For this article, I assume you are familiar with the syntax of regular expressions. If this is not the case, you might want to brush up by reading the article ".NET Framework Regular Expressions" at msdn.microsoft.com/library/hs600312, or for a deeper dive, read Jeffrey Friedl's excellent reference book, "Mastering Regular Expressions 3rd Edition" (O'Reilly, 2006).

Backtracking: The Root of the Problem

There are essentially two different types of regular expression engines: Deterministic Finite Automaton (DFA) engines and Non-deterministic Finite Automaton (NFA) engines. A complete analysis of the differences between these two engine types is beyond the scope of this article; we only need to focus on two facts:

1. NFA engines are backtracking engines. Unlike DFAs, which evaluate each character in an input string at most one time, NFA engines can evaluate each character in an input string multiple times. (I'll later demonstrate how this backtracking evaluation algorithm works.) The backtracking approach has benefits, in that these engines can process more-complex regular expressions, such as those containing backreferences or capturing parentheses. It also has drawbacks, in that their processing time can far exceed that of DFAs.
2. The Microsoft .NET Framework System.Text.RegularExpression classes use NFA engines.

One important side effect of backtracking is that while the regex engine can fairly quickly confirm a positive match (that is, an input string does match a given regex), confirming a negative match (the input string does not match the regex) can take quite a bit longer. In fact, the engine must confirm that none of the possible "paths" through the input string match the regex, which means that all paths have to be tested.

With a simple non-grouping regular expression, the time spent to confirm negative matches is not a huge problem. For example, assume that the regular expression to be matched against is:

```
^\d+$
```

This is a fairly simple regex that matches if the entire input string is made up of only numeric characters. The ^ and $ characters represent the beginning and end of the string respectively, the expression \d represents a numeric character, and + indicates that one or more characters will match. Let's test this expression using 123456X as an input string.

This input string is obviously not a match, because X is not a numeric character. But how many paths would the sample regex have to evaluate to come to this conclusion? It would start its evaluation at the beginning of the string and see that the character 1 is a valid numeric character and matches the regex. It would then move on to the character 2, which also would match. So the regex has matched the string 12 at this point. Next it would try 3 (and match 123), and so on until it got to X, which would not match.

However, because our engine is a backtracking NFA engine, it does not give up at this point. Instead, it backs up from its current match (123456) to its last known good match (12345) and tries again from there. Because the next character after 5 is not the end of the string, the regex is not a match, and it backs up to its previous last known good match (1234) and tries again. This proceeds all the way until the engine gets back to its first match (1) and finds that the character after 1 is not the end of the string. At this point the regex gives up; no match has been found.

All in all, the engine evaluated six paths: 123456, 12345, 1234, 123, 12 and 1. If the input string had been one character longer, the engine would have evaluated one more path. So this regular expression is a linear algorithm against the length of the string and is not at risk of causing a DoS. A System.Text.RegularExpressions.Regex object using ^\d+$ for its pattern is fast enough to tear through even enormous input strings (more than 10,000 characters) virtually instantly.

Any regular expression containing a grouping expression with repetition that is itself repeated is going to be vulnerable.

Now let's change the regular expression to group on the numeric characters:

```
^(\d+)$
```

This does not substantially change the outcome of the evaluations; it simply lets the developer access any match as a captured group. (This technique can be useful in more complicated regular expressions where you might want to apply repetition operators, but in this particular case it has no value.) Adding grouping parentheses in this case does not substantially change the expression's execution speed, either. Testing the pattern against the input 123456X still causes the engine to evaluate just six different paths. However, the situation is dramatically different if we make one more tiny change to the regex:

```
^(\d+)+$
```

The extra + character after the group expression (\d+) tells the regex engine to match any number of captured groups. The engine proceeds as before, getting to 123456 before backtracking to 12345.

Here is where things get "interesting" (as in horribly dangerous). Instead of just checking that the next character after 5 is not the end of the string, the engine treats the next character, 6, as a new capture group and starts rechecking from there. Once that route fails, it backs up to 1234 and then tries 56 as a separate capture group, then 5 and 6 each as separate capture groups. The end result is that the engine actually ends up evaluating 32 different paths.

If we now add just one more numeric character to the evaluation string, the engine will have to evaluate 64 paths—twice as many—to determine that it's not a match. This is an exponential increase in the amount of work being performed by the regex engine. An attacker could provide a relatively short input string—30 characters or so—and force the engine to process hundreds of millions of paths, tying it up for hours or days.

Airing Your Dirty Laundry

It's bad enough when an application has DoS-able exponential regexes tucked away in server-side code. It's even worse when an application advertises its vulnerabilities in client-side code. Many of the ASP.NET validator controls derived from System.Web.UI.WebControls.BaseValidator, including RegularExpressionValidator, will automatically execute the same validation logic on the client in JavaScript as they do on the server in .NET code.

Most of the time, this is a good thing. It's good to save the user the round-trip time of a form submission to the server just to have it rejected by a validator because the user mistyped an input field. It's good to save the server the processing time, too. However, if the application is using a bad regex in its server code, that bad regex is also going to be used in its client code, and now it will be extremely easy for an attacker to find that regex and develop an attack string for it.

For example, say I create a new Web form and add a TextBox and a RegularExpressionValidator to that form. I set the validator's ControlToValidate property to the name of the text box and set its ValidationExpression to one of the bad regexes I've discussed:

```
this.RegularExpressionValidator1.ControlToValidate = "TextBox1";
this.RegularExpressionValidator1.ValidationExpression = @"^(\d+)+$";
```

If I now open this page in a browser and view its source, I see the following JavaScript code close to the bottom of the page:

```
<scripttype="text/javascript">
//<![CDATA[
var RegularExpressionValidator1 = document.all ?
  document.all["RegularExpressionValidator1"] :
  document.getElementById("RegularExpressionValidator1");
RegularExpressionValidator1.controltovalidate = "TextBox1";
RegularExpressionValidator1.validationexpression = "^(\\d+)+$";
//]]>
</script>
```

There it is, for the whole world to see: the exponential regex in plain sight on the last line of the script block.

More Problem Patterns

Of course, ^(\d+)+$ is not the only bad regular expression in the world. Basically, any regular expression containing a grouping expression with repetition that is itself repeated is going to be vulnerable. This includes regexes such as:

```
^(\d+)*$
^(\d*)*$
^(\d+|\s+)*$
```

In addition, any group containing alternation where the alternate subexpressions overlap one another is also vulnerable:

```
^(\d|\d\d)+$
^(\d|\d?)+$
```

If you saw an expression like the previous sample in your code now, you'd probably be able to identify it as vulnerable just from looking at it. But you might miss a vulnerability in a longer, more complicated (and more realistic) expression:

```
^([0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*@(([0-9a-zA-Z])+([-\w]*[0-9a-zA-
Z])*\.)+[a-zA-Z]{2,9})$
```

This is a regular expression found on the Regular Expression Library Web site (regexlib.com) that is intended to be used to validate an e-mail address. However, it's also vulnerable to attack. You might find this vulnerability through manual code inspection, or you might not. A better technique to find these problems is required, and that's what I'm going to discuss next.

Finding Bad Regexes in Your Code

Ideally, there would be a way to find exponential regexes in your code at compile time and warn you about them. Presumably, in order to parse a regex string and analyze it for potential weaknesses, you would need yet another regex. At this point, I feel like a regex addict: "I don't need help, I just need more regexes!" Sadly, my regex skills are not up to the task of writing a regex to analyze regexes. If you believe you have working code for this, send it to me and I'll be happy to give you credit in next month's Security Briefs column. In the meantime, because I don't have a way to detect bad regexes at compile time, I'll do the next best thing: I'll write a regex fuzzer.

Fuzzing is the process of supplying random, malformed data to an application's inputs to try to make it fail. The more fuzzing iterations you run, the better the chance you'll find a bug, so it's common for teams to run thousands or millions of iterations per input. Microsoft security teams have found this to be an incredibly effective way to find bugs, so the Security Development Lifecycle team has made fuzzing a requirement for all product and service teams.

For my fuzzer, I want to fuzz random input strings to my regular expression. I'll start by defining a const string for my regex, a testInputValue method that checks the regex and a runTest method that will collect random input strings to feed to testInputValue.

```
const string regexToTest = @"^(\d+)+$";

static void testInputValue(string inputToTest)
{
  System.Text.RegularExpressions.Regex.Match(inputToTest, regexToTest);
}

void runTest()
{
  string[] inputsToTest = {};

  foreach (string inputToTest in inputsToTest)
  testInputValue(inputToTest);
}
```

Note that there's no code yet to generate the fuzzed input values; I'll get to that shortly. Also note that the code doesn't bother to check the return value from Regex.Match. This is because I don't actually care whether the input matches the pattern or not. All I care about in this situation is whether the regex engine takes too long to decide whether the input matches.

Normally fuzzers are used to try to find exploitable privilege elevation vulnerabilities, but again, in this case, I'm only interested in finding DoS vulnerabilities. I can't simply feed my test application data to see if it crashes; I have to be able to detect whether it's locked up. Although it may not be the most scientific method, I can accomplish this effectively by running each regex test sequentially on a separate worker thread and setting a timeout value for that thread's completion. If the thread does not complete its processing within a reasonable amount of time, say five seconds to test a single input, we assume that the regular expression has been DoS'd. I'll add a ManualResetEvent and modify the testInputValue and runTest methods accordingly, as shown in **Figure 1**.

Figure 1 **Testing Using Separate Worker Threads**

```
static ManualResetEvent threadComplete = new ManualResetEvent(false);

static void testInputValue(object inputToTest)
{
  System.Text.RegularExpressions.Regex.Match((string)inputToTest,
    regexToTest);
  threadComplete.Set();
}

void runTest()
{
  string[] inputsToTest = {};

  foreach (string inputToTest in inputsToTest)
  {
    Thread thread = new Thread(testInputValue);
    thread.Start(inputToTest);

    if (!threadComplete.WaitOne(5000))
    {
      Console.WriteLine("Regex exceeded time limit for input " +
        inputToTest);
      return;
    }

    threadComplete.Reset();
  }

  Console.WriteLine("All tests succeeded within the time limit.");
}
```

Now it's time to generate the input values. This is actually more difficult than it sounds. If I just generate completely random data, it's unlikely any of it would match enough of the regex to reveal a vulnerability. For example, if I test the regex ^(\d+)+$ with the input XdO(*iLy@Lm4p$, the regex will instantly not match and the problem will remain hidden. I need to generate input that's fairly close to what the application expects for the test to be useful, and for that I need a way to generate random data that matches a given regex.

Data Generation Plans to the Rescue

Fortunately, there is a feature in Visual Studio Database Projects that can do just that: the data generation plan. If you're using Visual Studio Team Suite, you also have access to this feature. Data generation plans are used to quickly fill databases with test data. They can fill tables with random strings, or numeric values or (luckily for us) strings matching specified regular expressions.

You first need to create a table in a SQL Server 2005 or 2008 database into which you can generate test data. Once that's done, come back into Visual Studio and create a new SQL Server Database project. Edit the database project properties to provide it with a connection string to your database. Once you've entered a connection string and tested it to make sure it works, return to the Solution Explorer and add a new Data Generation Plan item to the project. At this point, you should see something like **Figure 2**.

Now choose the table and column you want to fill with fuzzer input data. In the table section, set the number of test values to be generated (the Rows to Insert column). I wrote earlier that fuzzers

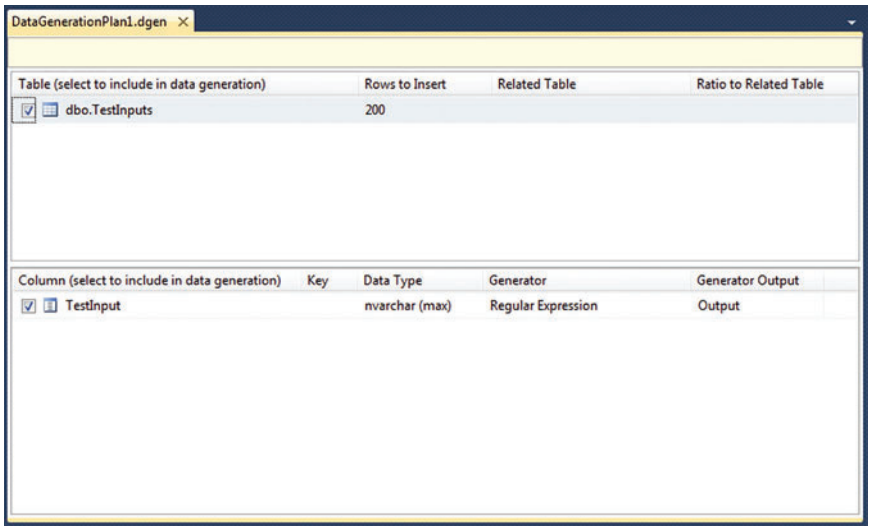

DataGenerationPlan1.dgen

Table (select to include in data generation)	Rows to Insert	Related Table	Ratio to Related Table
☑ dbo.TestInputs	200		

Column (select to include in data generation)	Key	Data Type	Generator	Generator Output
☑ TestInput		nvarchar (max)	Regular Expression	Output

Figure 2 **A Data Generation Plan Item in Visual Studio**

generally test hundreds of thousands or millions of iterations to try to find problems. Though I usually approve of this level of rigor, it's overkill for the purposes of this regex fuzzer. If a regex is going to lock up, it's going to do so within a couple hundred test iterations. I suggest setting the Rows to Insert value to 200, but if you want to test more, please feel free.

In the column section, now set the Generator to Regular Expression and enter the regex pattern value you want to test as the value of the Expression property in the column's Properties tab. It's important to note that the Expression property doesn't support every legal regular expression character. You can't enter the beginning- and end-of-line anchors ^ and $ (or more accurately, you can enter them, but the generator will generate a literal ^ or $ character in the test input). Just leave these characters out. You'll find a full list of operators supported by the Regular Expression Generator at msdn.microsoft.com/library/aa833197(VS.80).

In the worst-case scenario, the processing time is actually exponential to the number of characters in the input string.

A bigger problem is that the Expression property also doesn't support common shorthand notations such as \d for numeric digits or \w for word characters. If your regex uses these, you'll have to replace them with their character class equivalents: [0-9] instead of \d, [a-zA-Z0-9_] instead of \w and so on. If you need to replace \s (whitespace character), you can enter a literal space in its place.

Your last task in the database project is to actually fill the database with the test data according to your specifications. Do this by choosing the Data | DataGenerator | Generate Data menu item, or just press F5.

Adding the Attack

Back in the fuzzer code, modify the runTest method so it pulls the generated test data from the database. You might think you're done after this, but in fact there's one more important change to make. If you run the fuzzer now, even against a known bad regex such as ^(\d+)+$, it will fail to find any problems and report that all tests succeeded. This is because all the test data you've generated is a valid match for your regex.

Remember earlier I stated that NFA regex engines can fairly quickly confirm a positive match and that problems really only happen on negative matches. Furthermore, because of NFAs' backtracking nature, problems only occur when there are a large number of matching characters at the start of the input and the bad character appears at the end. If a bad character appeared at the front of the input string, the test would finish instantly.

The final change to make to the fuzzer code is to append bad characters onto the ends of the test inputs. Make a string array containing numeric, alphabetic, punctuation and whitespace characters:

```
string[] attackChars = { "0", "1", "9", "X", "x", "+",
"-", "@", "!", "(", ")", "[", "]", "\\", "/",
"?", "<", ">", ".", ",", ":", ";", " ", "" };
```

Now modify the code so each input string retrieved from the database is tested with each of these attack characters appended to it. So the first input string would be tested with a 0 character appended to it, then with a 1 character appended to it and so on. Once that input string has been tested with each of the attack characters, move to the next input string and test it with each of the attack characters.

```
foreach (string inputToTest in inputsToTest)
{
  foreach (string attackChar in attackChars)
  {
    Threadthread = new Thread(testInputValue);
    thread.Start(inputToTest + attackChar);
...
```

Now You Have Two Problems

There is a famous quote by ex-Netscape engineer Jamie Zawinski concerning regular expressions:

"Some people, when confronted with a problem, think, 'I know, I'll use regular expressions.' Now they have two problems."

While I am nowhere near as cynical about regexes as Mr. Zawinski, I will admit that it can be quite challenging just to write a correct regex, much less a correct regex that is secure against DoS attacks. I encourage you to examine all of your regexes for exponential complexity, and to use fuzzing techniques to verify your findings. ■

BRYAN SULLIVAN *is a security program manager for the Microsoft Security Development Lifecycle team, where he specializes in Web application and .NET security issues. He is the author of "Ajax Security" (Addison-Wesley, 2007).*

THANKS to the following technical experts for reviewing this article: *Barclay Hill, Michael Howard, Ron Petrusha and Justin van Patten*

TED NEWARD

Going NoSQL with MongoDB

Over the past decade or so, since the announcement of the Microsoft .NET Framework in 2000 and its first release in 2002, .NET developers have struggled to keep up with all the new things Microsoft has thrown at them. And as if that wasn't enough, "the community"—meaning both developers who use .NET on a daily basis and those who don't—has gone off and created a few more things to fill in holes that Microsoft doesn't cover—or just to create chaos and confusion (you pick).

One of those "new" things to emerge from the community from outside of the Microsoft *aegis* is the NoSQL movement, a group of developers who openly challenge the idea that all data is/will/must be stored in a relational database system of some form. Tables, rows, columns, primary keys, foreign key constraints, and arguments over nulls and whether a primary key should be a natural or unnatural one … is nothing sacred?

In this article and its successors, I'll examine one of the principal tools advocated by those in the NoSQL movement: MongoDB, whose name comes from "hu*mongo*us," according to the MongoDB Web site (and no, I'm not making that up). Most everything MongoDB-ish will be covered: installing, exploring and working with it from the .NET Framework, including the LINQ support offered; using it from other environments (desktop apps and Web apps and services); and how to set it up so the production Windows admins don't burn you in effigy.

Problem (or, Why Do I Care, Again?)

Before getting too deep into the details of MongoDB, it's fair to ask why any .NET Framework developers should sacrifice the next half-hour or so of their lives reading this article and following along on their laptops. After all, SQL Server comes in a free and redistributable Express edition that provides a lighter-weight data storage option than the traditional enterprise- or datacenter-bound relational database, and there are certainly plenty of tools and libraries available to provide easier access to it, including Microsoft's own LINQ and Entity Framework.

The problem is that the strength of the relational model—the relational model itself—is also its greatest weakness. Most developers, whether .NET, Java or something else entirely, can—after only a few years' experience—describe in painful detail how everything doesn't fit nicely into a tables/rows/columns "square" model. Trying to model hierarchical data can drive even the most experienced developer completely bonkers, so much so that Joe Celko wrote a book—"SQL for Smarties, Third Edition," (Morgan-Kaufmann, 2005)—entirely about the concept of modeling hierarchical data in a relational model. And if you add to this the basic "given" that relational databases assume an inflexible structure to the data—the database schema—trying to support ad hoc "additionals" to the data becomes awkward. (Quick, show of hands: How many of you out there work with databases that have a Notes column, or even better, Note1, Note2, Note3 …?)

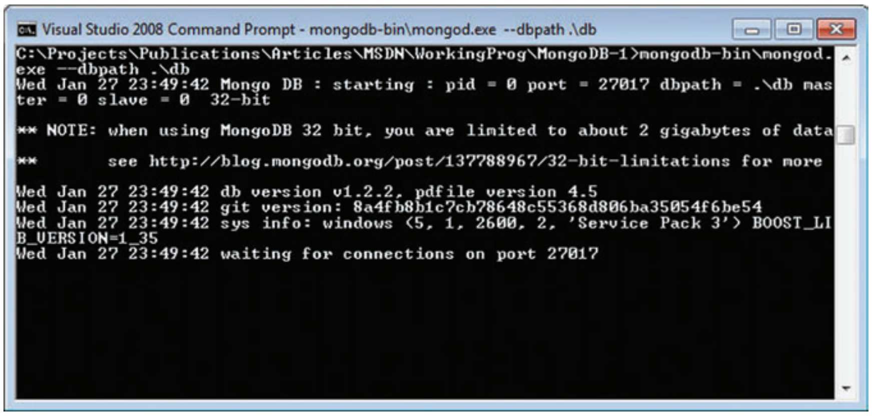
Nobody within the NoSQL movement is going to suggest that the relational model doesn't have its strengths or that the relational database is going to go away, but a basic fact of developer life in the past two decades is that developers have frequently stored data in relational databases that isn't inherently (or sometimes even remotely) relational in nature.

The problem is that the strength of the relational model—the relational model itself—is also its greatest weakness.

The document-oriented database stores "documents" (tightly knit collections of data that are generally not connected to other data elements in the system) instead of "relations." For example, blog entries in a blog system are entirely unconnected to one another, and even when one does reference another, most often the connection is through a hyperlink that is intended to be dereferenced by the user's browser, not internally. Comments on that blog entry are entirely scoped to that blog entry, and rarely do users ever want to see the aggregation of all comments, regardless of the entry they comment on.

Moreover, document-oriented databases tend to excel in high-performance or high-concurrency environments; MongoDB is particularly geared toward high performance, whereas a close cousin of it, CouchDB, aims more at high-concurrency scenarios. Both forgo any sort of multi-object transaction support, meaning that although they support concurrent modification of a single object in a database, any attempt to modify more than one at a time leaves a small window of time where those modifications can be seen "in passing." Documents are updated atomically, but there's no concept of a transaction that spans multiple-document updates.

Code download available at code.msdn.microsoft.com/mag201005WorkProg.



```
Visual Studio 2008 Command Prompt - mongodb-bin\mongod.exe --dbpath .\db
C:\Projects\Publications\Articles\MSDN\WorkingProg\MongoDB-1>mongodb-bin\mongod.
exe --dbpath .\db
Wed Jan 27 23:49:42 Mongo DB : starting : pid = 0 port = 27017 dbpath = .\db mas
ter = 0 slave = 0  32-bit

** NOTE: when using MongoDB 32 bit, you are limited to about 2 gigabytes of data

**       see http://blog.mongodb.org/post/137788967/32-bit-limitations for more

Wed Jan 27 23:49:42 db version v1.2.2, pdfile version 4.5
Wed Jan 27 23:49:42 git version: 8a4fb8b1c7cb78648c55368d806ba35054f6be54
Wed Jan 27 23:49:42 sys info: windows (5, 1, 2600, 2, 'Service Pack 3') BOOST_LI
B_VERSION=1_35
Wed Jan 27 23:49:42 waiting for connections on port 27017
```

Figure 1 **Firing up Mongod.exe to Verify Successful Installation**

This doesn't mean that MongoDB doesn't have any durability—it just means that the MongoDB instance isn't going to survive a power failure as well as a SQL Server instance does. Systems requiring full atomicity, consistency, isolation and durability (ACID) semantics are better off with traditional relational database systems, so mission-critical data most likely won't be seeing the inside of a MongoDB instance any time soon, except perhaps as replicated or cached data living on a Web server.

Document-oriented databases tend to excel in high-performance and high-concurrency environments; MongoDB is particularly geared toward high performance.

In general, MongoDB will work well for applications and components that need to store data that can be accessed quickly and is used often. Web site analytics, user preferences and settings—and any sort of system in which the data isn't fully structured or needs to be structurally flexible—are natural candidates for MongoDB. This doesn't mean that MongoDB isn't fully prepared to be a primary data store for operational data; it just means that MongoDB works well in areas that the traditional RDBMS doesn't, as well as a number of areas that could be served by either.

Getting Started

As mentioned earlier, MongoDB is an open-source software package easily downloaded from the MongoDB Web site, mongodb.com. Opening the Web site in a browser should be sufficient to find the links to the Windows downloadable binary bundle; look at the right-hand side of the page for the Downloads link. Or, if you prefer direct links, use mongodb.org/display/DOCS/Downloads. As of this writing, the stable version is the 1.2.4 release. It's nothing more than a .zip file bundle, so installing it is, comparatively speaking, ridiculously easy: just unzip the contents anywhere desired.

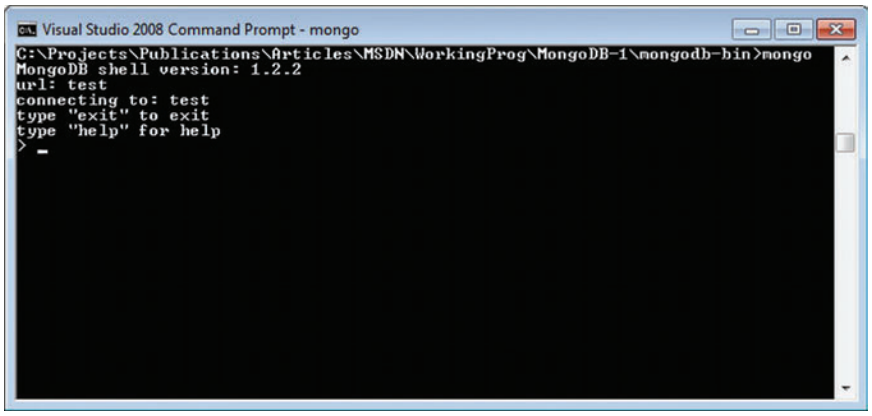
Seriously. That's it.

The .zip file explodes into three directories: bin, include and lib. The only directory of interest is bin, which contains eight executables. No other binary (or runtime) dependencies are necessary, and in fact, only two of those executables are of interest at the moment. These are mongod.exe, the MongoDB database process itself, and mongo.exe, the command-line shell client, which is typically used in the same manner as the old isql.exe SQL Server command-line shell client—to make sure things are installed correctly and working; browse the data directly; and perform administrative tasks.

Verifying that everything installed correctly is as easy as firing up mongod from a command-line client. By default, MongoDB wants to store data in the default file system path, c:\data\db, but this is configurable with a text file passed by name on the command line via --config. Assuming a subdirectory named db exists wherever mongod will be launched, verifying that everything is kosher is as easy as what you see in **Figure 1**.

If the directory doesn't exist, MongoDB will not create it. Note that on my Windows 7 box, when MongoDB is launched, the usual "This application wants to open a port" dialog box pops up. Make sure the port (27017 by default) is accessible, or connecting to it will be … awkward, at best. (More on this in a subsequent article, when I discuss putting MongoDB into a production environment.)

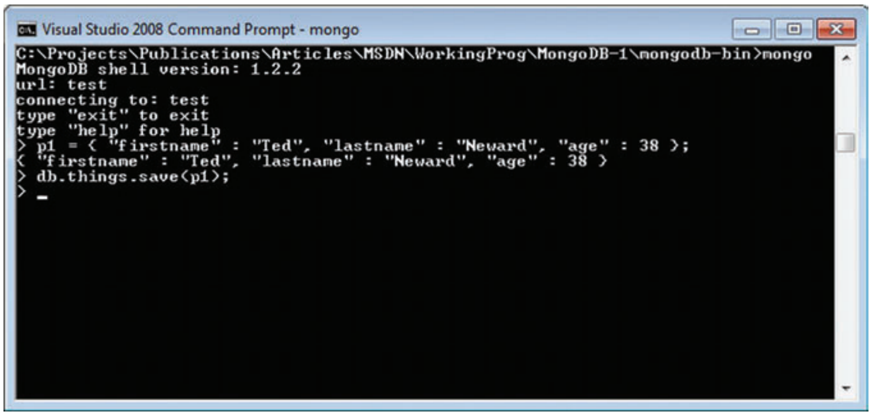
Once the server is running, connecting to it with the shell is just as trivial—the mongo.exe application launches a command-line environment that allows direct interaction with the server, as shown in **Figure 2**.



```
Visual Studio 2008 Command Prompt - mongo
C:\Projects\Publications\Articles\MSDN\WorkingProg\MongoDB-1\mongodb-bin>mongo
MongoDB shell version: 1.2.2
url: test
connecting to: test
type "exit" to exit
type "help" for help
> _
```

Figure 2 **Mongo.exe Launches a Command-Line Environment that Allows Direct Interaction with the Server**

By default, the shell connects to the "test" database. Because the goal here is just to verify that everything is working, test is fine. Of course, from here it's fairly easy to create some sample data to work with MongoDB, such as a quick object that describes a person. It's a quick glimpse into how MongoDB views data to boot, as we see in **Figure 3**.



```
C:\Projects\Publications\Articles\MSDN\WorkingProg\MongoDB-1\mongodb-bin>mongo
MongoDB shell version: 1.2.2
url: test
connecting to: test
type "exit" to exit
type "help" for help
> p1 = { "firstname" : "Ted", "lastname" : "Neward", "age" : 38 };
{ "firstname" : "Ted", "lastname" : "Neward", "age" : 38 }
> db.things.save(p1);
> _
```

Figure 3 **Creating Sample Data**

Essentially, MongoDB uses JavaScript Object Notation (JSON) as its data notation, which explains both its flexibility and the manner in which clients will interact with it. Internally, MongoDB stores things in BSON, a binary superset of JSON, for easier storage and indexing. JSON remains MongoDB's preferred input/output format, however, and is usually the documented format used across the MongoDB Web site and wiki. If you're not familiar with JSON, it's a good idea to brush up on it before getting heavily into MongoDB. Meanwhile, just for grins, peer into the directory in which mongod is storing data and you'll see that a couple of "test"-named files have shown up.

In general, MongoDB will work well for applications and components that need to store data that can be accessed quickly and is used often.

Enough playing—time to write some code. Quitting the shell is as easy as typing "exit," and shutting the server down requires only a Ctrl+C in the window or closing it; the server captures the close signal and shuts everything down properly before exiting the process.

MongoDB's server (and the shell, though it's not as much of an issue) is written as a native C++ application—remember those?—so accessing it requires some kind of .NET Framework driver that knows how to connect over the open socket to feed it commands and data. The MongoDB distribution doesn't have a .NET Framework driver bundled with it, but fortunately the community has provided one, where "the community" in this case is a developer by the name of Sam Corder, who has built a .NET Framework driver and LINQ support for accessing MongoDB. His work is available in both source and binary form, from github.com/samus/mongodb-csharp. Download either the binaries on that page (look in the upper-right corner) or the sources and build it. Either way, the result is two assemblies: MongoDB.Driver.dll and MongoDB.Linq.dll. A quick Add Reference to the References node of the project, and the .NET Framework is ready to rock.

Writing Code

Fundamentally, opening a connection to a running MongoDB server is not much different from opening a connection to any other database, as shown in **Figure 4**.

Discovering the object created earlier isn't hard, just … different … from what .NET Framework developers have used before (see **Figure 5**).

If this looks a bit overwhelming, relax—it's written out "the long way" because MongoDB stores things differently than traditional databases.

For starters, remember that the data inserted earlier had three fields on it—firstname, lastname and age, and any of these are elements by which the data can be retrieved. But more importantly, the line that stored them, tossed off rather cavalierly, was "test.things.save()"—which implies that the data is being stored in something called "things." In MongoDB terminology, "things" is a collection, and implicitly all data is stored in a collection. Collections in turn hold documents, which hold key/value pairs where the values can be additional collections. In this case, "things" is a collection stored inside of a database, which as mentioned earlier is the test database.

As a result, fetching the data means connecting first to the MongoDB server, then to the test database, then finding the collection "things." This is what the first four lines in **Figure 5** do—create a Mongo object that represents the connection, connects to the server, connects to the test database and then obtains the "things" collection.

The MongoDB distribution doesn't have a .NET Framework driver bundled with it, but fortunately the community has provided one.

Once the collection is returned, the code can issue a query to find a single document via the FindOne call. But as with all databases, the client doesn't want to fetch every document in the collection and then find the one it's interested in—somehow, the query needs to be constrained. In MongoDB, this is done by creating a Document that contains the fields and the data to search for in

Figure 4 **Opening a Connection to a MongoDB Server**

```
using System;
using MongoDB.Driver;

namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Mongo db = new Mongo();
      db.Connect(); //Connect to localhost on the default port
      db.Disconnect();
    }
  }
}
```

those fields, a concept known as query by example, or QBE for short. Because the goal is to find the document containing a lastname field whose value is set to "Neward," a Document containing one lastname field and its value is created and passed in as the parameter to FindOne. If the query is successful, it returns another Document containing all the data in question (plus one more field); otherwise it returns null.

By the way, the short version of this description can be as terse as:

```
Document anotherResult =
  db["test"]["things"].FindOne(
    new Document().Append("lastname", "Neward"));
Console.WriteLine(anotherResult);
```

When run, not only do the original values sent in show up, but a new one appears as well, an _id field that contains an ObjectId object. This is the unique identifier for the object, and it was silently inserted by the database when the new data was stored. Any attempt to modify this object must preserve that field or the database will assume it's a new object being sent in. Typically, this is done by modifying the Document that was returned by the query:

```
anotherResult["age"] = 39;
things.Update(resultDoc);
Console.WriteLine(
  db["test"]["things"].FindOne(
    new Document().Append("lastname", "Neward")));
```

However, it's always possible to create a new Document instance and manually fill out the _id field to match the ObjectId, if that makes more sense:

```
Document ted = new Document();
ted["_id"] = new MongoDB.Driver.Oid("4b61494aff75000000002e77");
ted["firstname"] = "Ted";
ted["lastname"] = "Neward";
ted["age"] = 40;
things.Update(ted);
Console.WriteLine(
  db["test"]["things"].FindOne(
    new Document().Append("lastname", "Neward")));
```

Of course, if the _id is already known, that can be used as the query criteria, as well.

Notice that the Document is effectively untyped—almost anything can be stored in a field by any name, including some core .NET Framework value types, such as DateTime. Technically, as mentioned, MongoDB stores BSON data, which includes some extensions to traditional JSON types (string, integer, Boolean, double and null—though nulls are only allowed on objects, not in collections) such as the aforementioned ObjectId, binary data, regular expressions and embedded JavaScript code. For the moment, we'll leave the latter two alone—the fact that BSON can store binary data means that anything that can be reduced to a byte array can be stored, which effectively means that MongoDB can store anything, though it might not be able to query into that binary blob.

Figure 5 **Discovering a Created Mongo Object**

```
using System;
using MongoDB.Driver;

namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Mongo db = new Mongo();
      db.Connect(); //Connect to localhost on the default port.
      Database test = db.getDB("test");
      IMongoCollection things = test.GetCollection("things");
      Document queryDoc = new Document();
      queryDoc.Append("lastname", "Neward");
      Document resultDoc = things.FindOne(queryDoc);
      Console.WriteLine(resultDoc);
      db.Disconnect();
    }
  }
}
```

The fact that BSON can store binary data means that anything that can be reduced to a byte array can be stored, which effectively means that MongoDB can store anything.

Not Dead (or Done) Yet!

There's much more to discuss about MongoDB, including LINQ support; doing more complex server-side queries that exceed the simple QBE-style query capabilities shown so far; and getting MongoDB to live happily in a production server farm. But for now, this article and careful examination of IntelliSense should be enough to get the working programmer started.

By the way, 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 .NET Framework and Java platform systems. He has written more than 100 articles, is a C# MVP, INETA speaker and has authored and coauthored a dozen books, including the forthcoming "Professional F# 2.0" (Wrox). He consults and mentors regularly—reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.*

THANKS to the following technical experts for reviewing this article: *Kyle Banker and Sam Corder*

Thinking Outside the Grid

The Canvas is one of several layout options available in Windows Presentation Foundation (WPF) and Silverlight, and it's the one most firmly rooted in tradition. When filling the Canvas with children, you position each child by specifying coordinates using the Canvas.Left and Canvas.Top attached properties. This is quite a different paradigm from the other panels, which arrange child elements based on simple algorithms without any need for the programmer to figure out the actual locations.

When you hear the word "canvas," you probably think about painting and drawing. For that reason, perhaps, programmers using WPF and Silverlight tend to relegate the Canvas to the display of vector graphics. Yet, when you use the Canvas to display Line, Polyline, Polygon and Path elements, the elements themselves include coordinate points that position them within the Canvas. As a result, you don't need to bother with the Canvas.Left and Canvas.Top attached properties.

So why use a Canvas if you don't need the attached properties it provides? Is there a better approach?

Canvas vs. Grid

Over the years, I have increasingly tended to reject the Canvas for displaying vector graphics, gravitating instead toward the use of a single-cell Grid. A single-cell Grid is just like a regular Grid except without any row or column definitions. If the Grid has only one cell, you can put multiple elements into the Grid cell and you don't use any of the Grid's attached properties to indicate rows or columns.

Initially, using a Canvas or a single-cell Grid seems very similar. Regardless which one you use for vector graphics, Line, Polyline, Polygon and Path elements will be positioned relative to the upper-left corner of the container based on their coordinate points.

The difference between the Canvas and the single-cell Grid is in how the container appears to the rest of the layout system. WPF and Silverlight incorporate a two-pass, top-down layout where every element interrogates the size of its children and is then responsible for arranging its children relative to itself. Within this layout system, the Canvas and the single-cell Grid are very different:

- To its children, the Grid has the same dimensions as the dimensions of its own parent. These are usually finite dimensions, but the Canvas always appears to have infinite dimensions to its children.
- The Grid reports the composite size of its children to its parent. However, the Canvas always has an apparent size of zero, regardless of the children it contains.

Suppose you have a bunch of Polygon elements that form some kind of cartoon-like vector graphics image. If you put all these Polygon elements in a single-cell Grid, the size of the Grid is based on the maximum horizontal and vertical coordinates of the polygons. The Grid can then be treated as a normal finite-sized element within the layout system because its size properly reflects the size of the composite image. (Actually, this works correctly only if the upper-left corner of the image is at the point (0, 0), and there are no negative coordinates.)

Put all those polygons in a Canvas, however, and the Canvas reports to the layout system that it has a size of zero. In general, when integrating a composite vector graphics image into your application, you almost certainly want the behavior of the single-cell Grid rather than the Canvas.

So is the Canvas entirely useless? Not at all. The trick is to use the peculiarities of the Canvas to your advantage. In a very real sense, the Canvas doesn't participate in layout. Hence, you can use it whenever you need to transcend layout—to display graphics that break the bounds of the layout system and float outside it. By default the Canvas doesn't clip its children, so even if it is very small, it can still host children outside its bounds. The Canvas is more of a reference point for displaying elements or graphics than it is a container.

The Canvas is great for techniques I have come to regard as "thinking outside the Grid." Although I'll be showing the code examples in Silverlight, you can use the same techniques in WPF. The downloadable source code that accompanies this article is a Visual Studio solution named ThinkingOutsideTheGrid, and you can play with the programs at charlespetzold.com/silverlight/ThinkingOutsideTheGrid.

Visual Linking of Controls

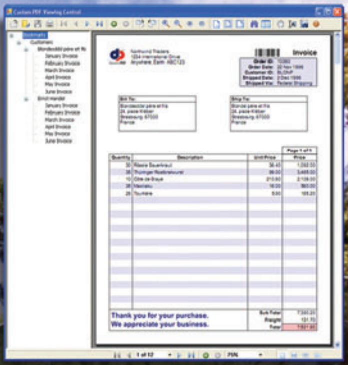
Suppose you have a bunch of controls in your Silverlight or WPF application and you need to provide some kind of visual link between two or more controls. Perhaps you want to draw a line from one control to another, and perhaps this line will cross other controls in between.

Certainly this line must react to changes in layout, perhaps as the window or page is resized by the user. Being informed when a layout is updated is an excellent application of the LayoutUpdated event—an event I never had occasion to use before exploring the problems I describe in this article. LayoutUpdated is defined by

Code download available at code.msdn.microsoft.com/mag201005UIFrontiers.

DynamicPDF

Dependable Developer Components



DynamicPDF Viewer

Our new, customizable DynamicPDF Viewer allows you to display PDF documents within any WinForm application. No longer rely on an external viewer for displaying your PDF documents. DynamicPDF Viewer utilizes the proven reliable and efficient Foxit PDF viewing engine and maximizes performance and compatibility with our other DynamicPDF products.

DynamicPDF Converter

Our DynamicPDF Converter library can efficiently convert over 30 document types (including HTML and all common Office file formats) to PDF. Events can be used to manage the action taken on a successful or failed conversion. It is highly intuitive and flexible and integrates well with our other DynamicPDF products.

DynamicPDF Rasterizer

Our DynamicPDF Rasterizer library can quickly convert PDF documents to over 10 common image formats including multi-page TIFF. Rasterizing form field values as well as annotations is fully supported. PDFs can also be rasterized to a System.Drawing.Bitmap class for further manipulation.

Try our three new products FREE today!

Fully functional and never expiring evaluation editions available at www.cete.com/download

To learn more about these or any of our other popular tools: DynamicPDF Generator, DynamicPDF Merger, DynamicPDF ReportWriter, DynamicPDF Suite, DynamicPDF WebCache or Firemail, visit us online.

ceTesoftware
INFINITE POSSIBILITIES

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.

www.cete.com
info@cete.com

800.631.5006
+1 410.772.8620

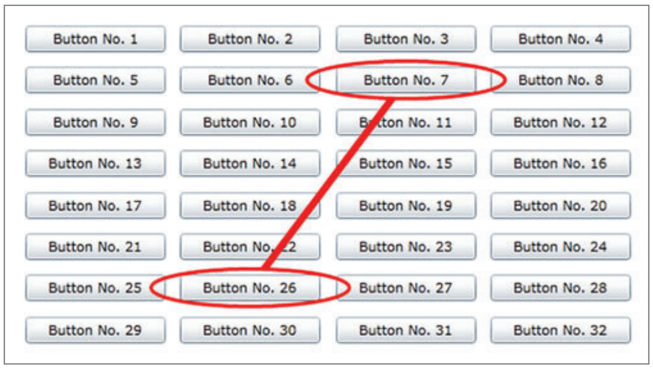


Figure 1 **The ConnectTheElements Display**

UIElement in WPF and by FrameworkElement in Silverlight. As the name suggests, the event is fired after a layout pass has rearranged elements on the screen.

When processing the LayoutUpdated event, you don't want to do anything that will cause another layout pass and get you embroiled in an infinite recursion. That's where the Canvas comes in handy: Because it always reports a zero size to its parent, you can alter elements in the Canvas without affecting layout.

The XAML file of the ConnectTheElements program is structured like this:

```
<UserControl … >
  <Grid ... >
    <local:SimpleUniformGrid … >
      <Button ... />
      <Button ... />
      ...
    </local:SimpleUniformGrid>

    <Canvas>
      <Path ... />
      <Path ... />
      <Path ... />
    </Canvas>
  </Grid>
</UserControl>
```

The Grid contains a SimpleUniformGrid that calculates the number of rows and columns to display its children based on its overall size and aspect ratio. As you change the size of the window, the number of rows and columns will change and cells will shift around. Of the 32 buttons in this SimpleUniformGrid, two of the buttons have names of btnA and btnB. The Canvas occupies the same area as the SimpleUniformGrid, but sits on top of it. This Canvas contains Path elements that the program uses to draw ellipses around the two named buttons and a line between them.

The code-behind file does all its work during the LayoutUpdated event. It needs to find the location of the two named buttons relative to the Canvas, which conveniently is also aligned with the SimpleUniformGrid, the Grid and MainPage itself.

To find a location of any element relative to any other element in the same visual tree, use the TransformToVisual method. This method is defined by the Visual class in WPF and by UIElement in Silverlight, but it works the same way in both environments. Suppose the element el1 is somewhere within the area occupied by el2. (In ConnectTheElements, el1 is a Button and el2 is MainPage.) This method call returns an object of type GeneralTransform, which is the abstract parent class to all the other graphics transform classes:

```
el1.TransformToVisual(el2)
```

You can't really do anything with GeneralTransform except call its Transform method, which transforms a point from one coordinate space to another.

Suppose you want to find the center of el1 but in el2's coordinate space. Here's the code:

```
Point el1Center = new Point(
  el1.ActualWidth / 2, el1.ActualHeight / 2);
Point centerInEl2 =
  el1.TransformToVisual(el2).Transform(el1Center);
```

If el2 is either the Canvas or aligned with the Canvas, you can then use that centerInEl2 point to set a graphic in the Canvas that will seemingly be positioned in the center of el1.

ConnectTheElements performs this transform in its WrapEllipseAroundElement method to draw ellipses around the two named buttons, and then calculates the coordinates of the line between the ellipses, based on an intersection of the line between the centers of the buttons. **Figure 1** shows the result.

If you try this program in WPF, change the SimpleUniformGrid to a WrapPanel for a more dynamic change in layout as you resize the program's window.

Tracking Sliders

Changing graphics and other visuals in response to changes in a scrollbar or slider is very basic, and in WPF and Silverlight you can do it in either code or a XAML binding. But what if you want to align the graphics exactly with the actual slider thumb?

This is the idea behind the TriangleAngles project, which I conceived as a type of interactive trigonometry demonstration. I arranged two sliders, one vertical and one horizontal, at right angles to each other. The two slider thumbs define the two vertices of a right triangle, as shown in **Figure 2**.

Notice how the semi-transparent triangle sits on top of the two sliders. As you move the slider thumbs, the sides of the triangle change size and proportion, as indicated by the inscribed angles and the labels on the vertical and horizontal legs.

This is obviously another job for a Canvas overlay, but with an added layer of complexity because the program needs to get access to the slider thumb. That slider thumb is part of a control template: the thumbs are assigned names within the template, but unfortunately these names can't be accessed outside the template.

Instead, the frequently essential VisualTreeHelper static class comes to the rescue. This class lets you walk (or rather climb) any visual tree in WPF or Silverlight through the GetParent, GetChildenCount and GetChild methods. To generalize the process of locating a specific type child, I wrote a little recursive generic method:

```
T FindChild<T>(DependencyObject parent)
  where T : DependencyObject
```

I call it like this:

```
Thumb vertThumb = FindChild<Thumb>(vertSlider);
Thumb horzThumb = FindChild<Thumb>(horzSlider);
```

At that point, I could use TransformToVisual on the two thumbs to obtain their coordinates relative to the Canvas overlay.

Well, it worked for one slider, but not the other, and it took me awhile to recall that the control template for the Slider contains *two* thumbs—one for the horizontal orientation and one for the vertical. Depending on the orientation set for the Slider, half the template has its Visibility property set to Collapsed. I added a second argument to the FindChild method called mustBeVisible and used that to abandon the search down any child branch where an element is not visible.

Setting HitTestVisible to false on the Polygon that forms the triangle helped prevent it from interfering with mouse input to the Slider thumb.

Scrolling Outside the ItemsControl

Suppose you're using an ItemsControl or a ListBox with a DataTemplate to display the objects in the control's collection. Can you include a Canvas in that DataTemplate so information concerning a particular item can be displayed outside the control, but seems to track the item as the control is scrolled?

I haven't found a good way to do precisely that. The big problem seems to be a clipping region imposed by the ScrollViewer. This ScrollViewer clips any Canvas that happens to dangle outside its boundary, and consequently anything on that Canvas.

However, with a little additional knowledge of the ItemsControl inner workings, you can do something close to what you want.

I think of this feature as a pop-out in that it's something that pertains to an item in an ItemsControl, but is actually popped out of the ItemsControl itself. The ItemsControlPopouts project demonstrates the technique. To provide something for the ItemsControl to display, I created a little database called ProduceItems.xml that resides in the Data subdirectory of ClientBin. ProduceItems consists of a number of elements with the tag name of ProduceItem, each of which contains a Name attribute, a Photo attribute referencing a bitmap picture of the item and an optional Message, which will be displayed "popped-out" of the ItemsControl. (The photos and other artwork are Microsoft Office clip art.)

The ProduceItem and ProduceItems classes provide code support for the XML file, and ProduceItemsPresenter reads the XML file and deserializes it into a ProduceItems object. This is set to the DataContext property of the visual tree that contains the ScrollViewer and ItemsControl. The ItemsControl contains a simple DataTemplate for displaying the items.

By now you may detect a bit of a problem. The program is effectively inserting business objects of type ProduceItem into the ItemsControl. Internally the ItemsControl is building a visual tree for each item based on the DataTemplate. To track the movement of these items you need access to that item's internal visual tree to figure out where exactly the items are relative to the rest of the program.

This information is available. ItemsControl defines a get-only property named ItemContainerGenerator that returns an object of type ItemContainerGenerator. This is the class responsible for generating the visual trees associated with each item in the ItemsControl, and it contains handy methods such as ContainerFromItem, which provides the container (which is actually a ContentPresenter) for each object in the control.

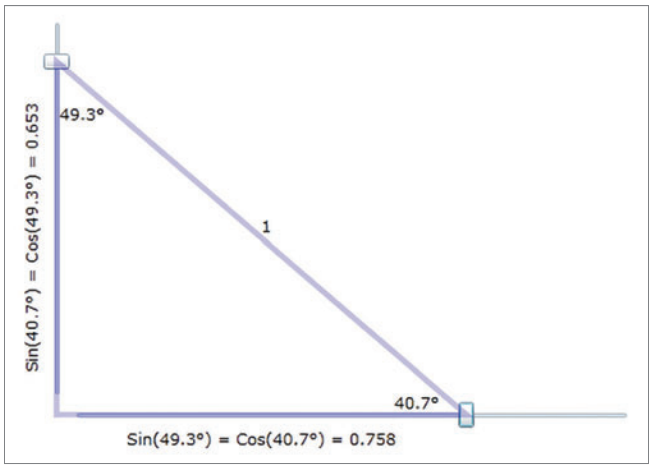

49.3°
1
40.7°
Sin(40.7°) = Cos(49.3°) = 0.653
Sin(49.3°) = Cos(40.7°) = 0.758

Figure 2 **The TriangleAngles Display**

The Canvas is great for techniques I have come to regard as "thinking outside the Grid."

Like the two other programs, the ItemsControlPopouts program covers the whole page with a Canvas. Once again the LayoutUpdated event allows the program to check whether something on the Canvas needs to be altered. The LayoutUpdated handler in this program enumerates through the ProduceItem objects in the ItemsControl and checks for a non-null and non-empty Message property. Each of these Message properties should correspond to an object of type PopOut in the Canvas. The PopOut is simply a small class that derives from ContentControl with a template to display a line and the message text. If the PopOut is not present, it's created and added to the Canvas. If it is present, it's simply reused.

The PopOut then must be positioned within the Canvas. The program obtains the container that corresponds to the data object and transforms its location relative to the canvas. If that location is between the top and bottom of the ScrollViewer, the PopOut has its Visibility property set to Visible. Otherwise the PopOut is hidden.

Breaking out of the Cell

WPF and Silverlight have certainly given the great gift of ease in layout. The Grid and other panels put elements neatly in cells and ensure that's where they stay. It would be a shame if you then assumed that convenience was a necessary limitation to the freedom to put elements wherever you want them. ■

CHARLES PETZOLD *is a longtime contributing editor to* MSDN Magazine. *His most recent book is "The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine" (Wiley, 2008). Petzold blogs on his Web site charlespetzold.com.*

THANKS to the following technical experts for reviewing this article: *Arathi Ramani and the WPF Layout Team*

VSLive!

JOIN US AT MICROSOFT HQ!

CALLING ALL DEVELOPERS!

Join Us August 2–6, in Redmond, Washington for Five Code-Packed Days on the Microsoft Campus!

VSLive! delivers an unbiased blend of practical, pragmatic and immediately-applicable knowledge, plus a glimpse of the future of technology. Interact one-on-one with the Microsoft development team and connect with industry experts during in-depth sessions on:

- Silverlight, WPF, XAML, VS10 Xaml designer, Blend, WCF RIA Services
- Web Forms, ASP.NET MVC, AJAX
- Cloud Computing
- Azure, Amazon, AppFabric, REST services, "Dallas", WCF, Windows Workflow
- SharePoint, Office
- SQL Server, BI, reporting, analysis, ADO.NET EF, ANDS, OData, Sync services
- Visual Studio features, TFS, languages, parallel extensions

And on top of all this great education, you'll have EXCLUSIVE access to check out Microsoft's corporate headquarters!

Register Now and Save Big! Go to vslive.com to Reserve Your Place on Campus!

Supported by:

Microsoft msdn Visual Studio MAGAZINE Microsoft Visual Studio

1105 MEDIA INC

AUGUST 2–6, 2010

ATTEND VSLIVE! REDMOND

AND LEARN ONE-ON-ONE WITH THE MIRCOSOFT DEVELOPMENT TEAM!

DETAILS AND REGISTRATION AT

VSLIVE.COM

USE PRIORITY CODE MSDN6

Platinum Sponsors

CodeFluent Entities

VERSANT

Fundamental Laws

Whenever I find a program or Web site that sucks, it's invariably because the developer or designer forgot the user's goals and started embellishing the program for its own sake. That violates Platt's 3 Laws of Software. Like the 3 Laws of Thermodynamics, they'll turn around and bite you if you do that. My laws are:

1 Your software has zero value in and of itself. Nothing. Nada. Zero point zero zero. The only value it ever has or ever will have is the degree to which it enhances the happiness of your user in some way.

2 Software can increase users' happiness in one of two ways. It can help a user accomplish a task that she wants done, such as paying her bills or writing this column. Microsoft Word is an example of this type of application. Or, it can place the user into a state he finds pleasurable. Games fall into this category, as does Skype, with which my daughters make video calls to their grandparents.

3 In neither of these cases do users want to think about your computer program. At all. Ever. In the former case, she wants to think about the problem she's solving: the wording of the document she's writing; or whether she has enough money to pay all her bills, and which unpaid creditor would hurt her the most if she doesn't. Her only goal is to finish the task quickly and successfully, so she can get on with her life, or at least with her next task. In the latter case, the user wants to enter that pleasurable state as quickly as possible and stay there as long as possible. Anything that delays the start of his pleasure, or distracts him from it while he's enjoying it, is even less welcome than the interruption of a work task. My parents want to see and talk with and gush over their grandchildren. Any attention that the program diverts to itself is a negative in either case.

To summarize: Users don't care about your program in and of itself. Never have, never will. Your mother might, because you wrote it and she loves you, and then again she might not; but no one else does. Users care only about their own productivity or their own pleasure.

Donald Norman discussed this phenomenon in his excellent book, "The Invisible Computer" (The MIT Press, 1999). But I see a better example in the role of sports referees. They set and enforce the context (rules of the game, operation of the program) for interactions between the parties (the two teams, the user and the business problem), while ideally taking no discernible part in it themselves. Retired NFL referee Jerry Markbreit writes of an early mentor, who said (emphasis added): "Gentlemen, this is the biggest game of the year. Maybe the biggest game in a hundred years. We must do an outstanding job today because *we don't want anyone to remember us*. We must work this game so expertly that, later, *it will seem as if we weren't even there*." That should be the goal of our programs as well.

To guide the industry toward this goal, I announce the first annual Plattski Awards, for the program or Web site that does the best job of implementing Platt's 3 Laws of Software, making its users happy while demanding as little attention as possible for itself. An example is the Carbonite automated backup program, about which I've written. I'd give it the nod over Mozy, its nearest competitor, because Carbonite requires somewhat less thinking and configuration.

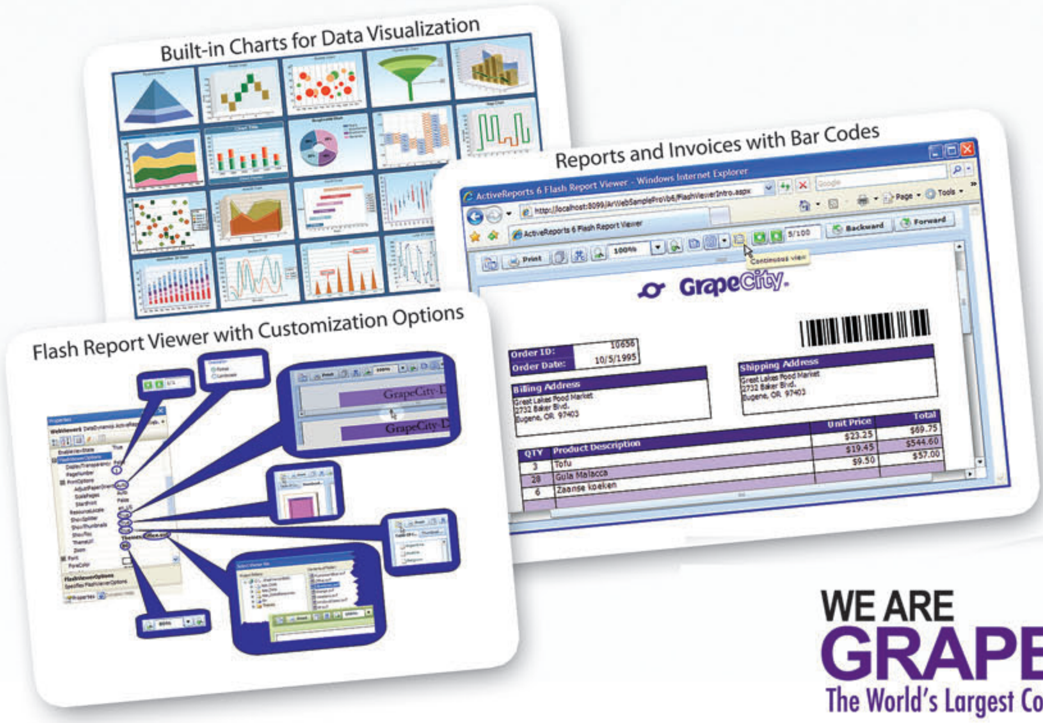
Your software has zero value in and of itself. Nothing. Nada. Zero point zero zero.

Go to Rollthunder.com and tell me about applications or Web sites that best accomplish this goal, or that fail most spectacularly. I'll give out gold(ish) and silver(y) medals to the best, and tin(plated) ones to the worst, in my own cranky and opinionated judgment. Do not send products: all testing will be conducted with publicly downloadable versions. The deadline for submissions is three months from the publication of this column. I can't wait to see what you come up with. ■

DAVID S. PLATT *teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" and "Introducing Microsoft .NET." 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.*

GrapeCity

WE ARE
REPORTING


Built-in Charts for Data Visualization
Reports and Invoices with Bar Codes
Flash Report Viewer with Customization Options
ActiveReports 6 Flash Report Viewer - Windows Internet Explorer
Print
Backward
Forward
GrapeCity

Order ID: 10656
Order Date: 10/5/1995

Billing Address
Great Lakes Food Market
2732 Baker Blvd.
Eugene, OR 97403

Shipping Address
Great Lakes Food Market
2732 Baker Blvd.
Eugene, OR 97403

QTY	Product Description	Unit Price	Total
3	Tofu	$23.25	$69.75
28	Gula Malacca	$19.45	$544.60
6	Zaanse koeken	$9.50	$57.00

WE ARE
GRAPECITY
The World's Largest Component Vendor

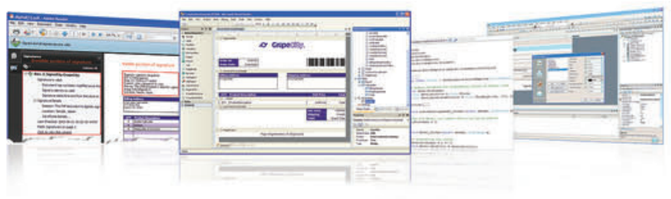
Integrate Business Intelligence/Reporting/Data Analysis into your .NET Applications using the NEW ActiveReports 6.

- ✔ 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
- ✔ Expert Technical Support and Extensive Samples and Documentation

PowerTools	SPREAD 5	ACTIVEREPORTS 6	ANALYSIS 1
Deep Functionality	100%	100%	100%
Award Winning	✓	✓	New
Category Leader	✓	✓	✓
Industry's Best Support	✓	✓	✓
Proven Track Record	✓	✓	✓

www.FarPointSpread.com & www.DataDynamics.com

ACTIVEREPORTS 6



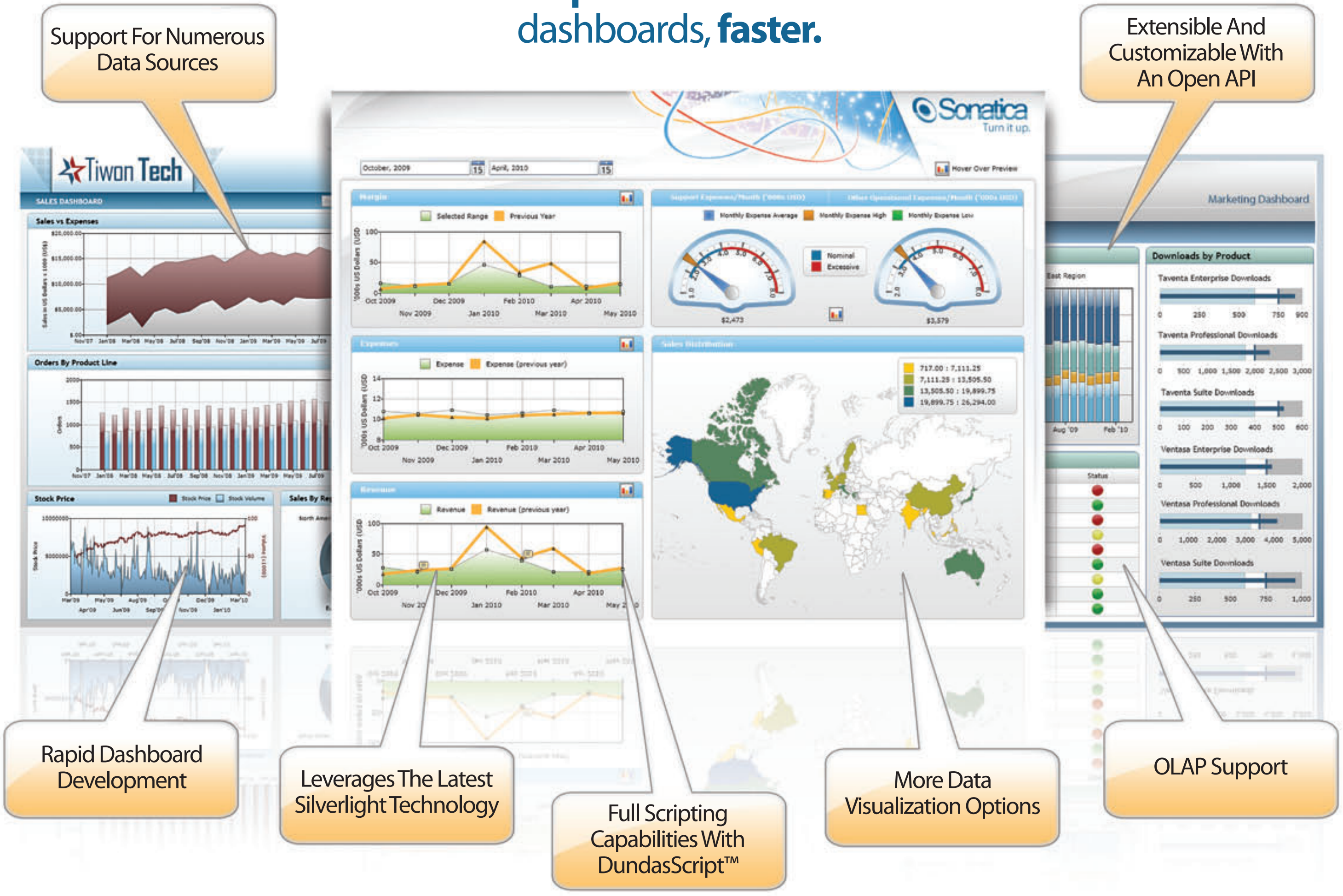
Download your free evaluation today at www.DataDynamics.com

Spreadsheets Reporting Analysis

Dundas Dashboard

The Dashboard Platform for Developers like you

Build more **powerful** and **effective** dashboards, **faster.**


Support For Numerous Data Sources
Extensible And Customizable With An Open API
Rapid Dashboard Development
Leverages The Latest Silverlight Technology
Full Scripting Capabilities With DundasScript™
More Data Visualization Options
OLAP Support

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