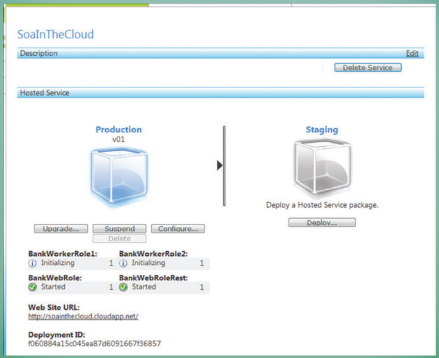
The Microsoft Journal for Developers

msdn® magazine



Deploy a Hosted Service package.

BankWorkerRole1: Initializing 1
BankWorkerRole2: Initializing 1
BankWebRole: Started 1
BankWebRoleRest: Started 1

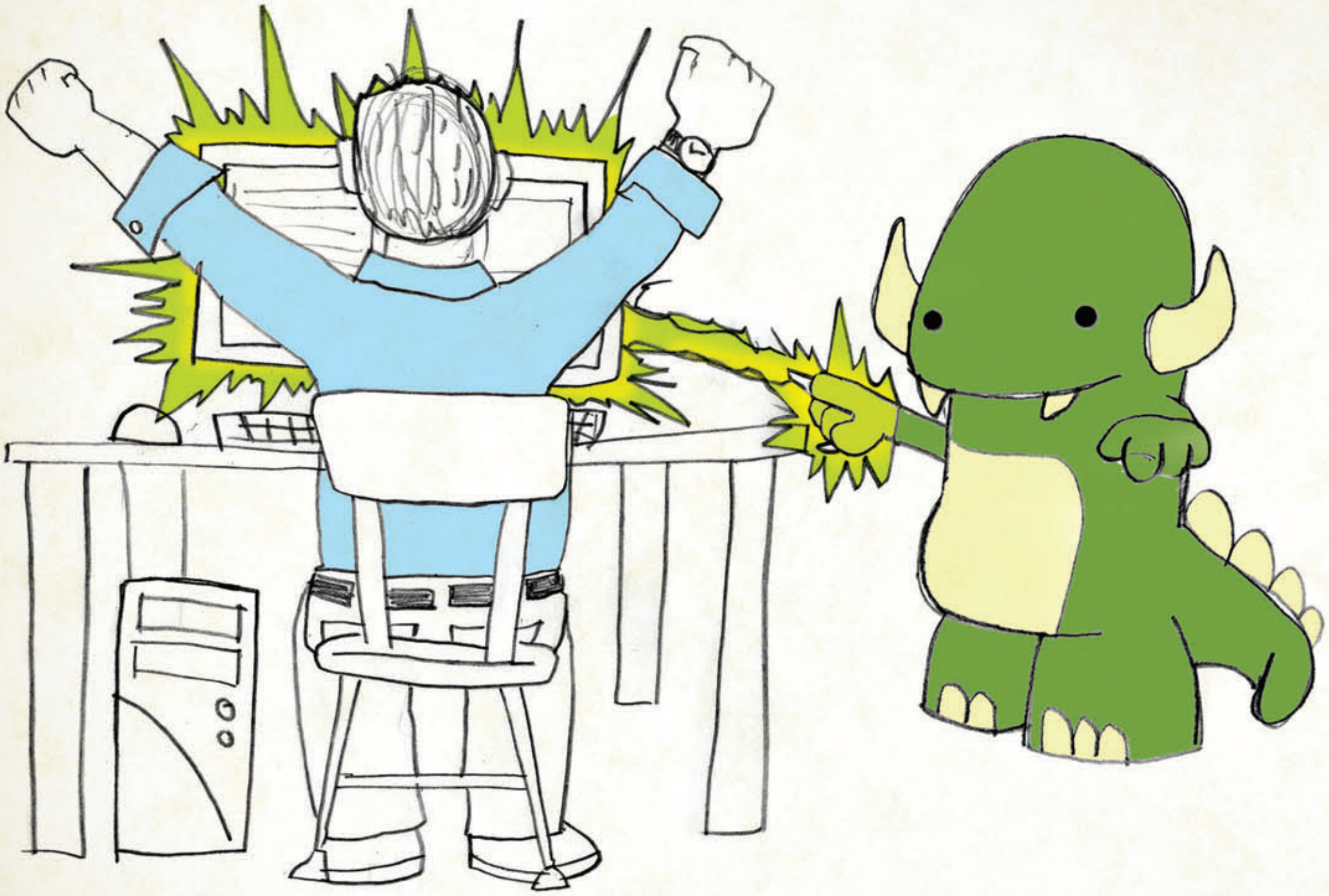
Web Site URL:
http://soainthecloud.cloudapp.net/

Deployment ID:
f060884a15c045ea87d6091667f36857

msdn

Microsoft®

JANUARY 2010 VOL 25 NO 1



GET THE POWER TO CREATE A KILLER APP

When an electrochemical reaction animated the dormant cells in a very powerful egg, Gort was hatched. With special powers and abilities to infuse ordinary applications with UIs that have extreme functionality, complete usability and the “wow-factor!”, Gort empowers Killer Apps. Go to infragistics.com/killerapps to find out how you can start creating your own Killer Apps.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111

Infragistics®

KILLER APPS. NO EXCUSES.

All cats can purr,

Syncfusion™
Deliver innovation with ease

www.syncfusion.com 1 888-9DOTNET

only lions roar.

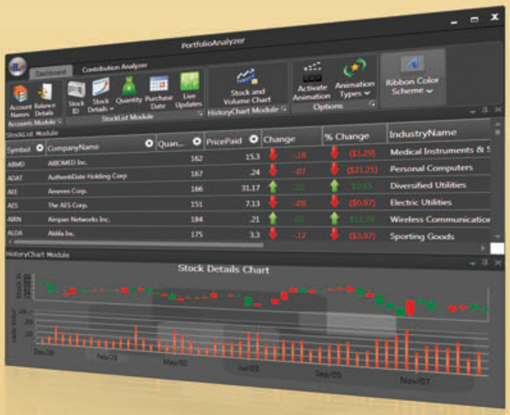
What does your grid do?

Although many grids share basic features, only Syncfusion's WPF grid has the power and performance to set your application apart. Our grid builds on years of Syncfusion's expertise with high-performance scenarios.

- Support for billions of rows
- Unmatched scrolling performance with large data sets
- Wide variety of styling options

Syncfusion Essential Grid WPF is the fastest WPF grid on the market.

Experience it for yourself at
www.syncfusion.com/wpf-gridperformance



Symbol	CompanyName	Quan...	PricePaid	Change	% Change	IndustryName
ABMD	ABIOMED Inc.	162	15.3	-.18	($1.29)	Medical Instruments & S
ADAT	AuthentiDate Holding Corp	167	.24	-.07	($21.21)	Personal Computers
AEE	Ameren Corp.	166	31.17	[illegible]	[illegible]	Diversified Utilities
AES	The AES Corp.	151	7.13	-.08	($0.97)	Electric Utilities
ARN	Airspan Networks Inc.	184	.21	[illegible]	[illegible]	Wireless Communicatio
ALDA	Aldila Inc.	175	3.3	-.12	($3.97)	Sporting Goods

ASP.NET MVC

ASP.NET

Windows Forms

Back Office

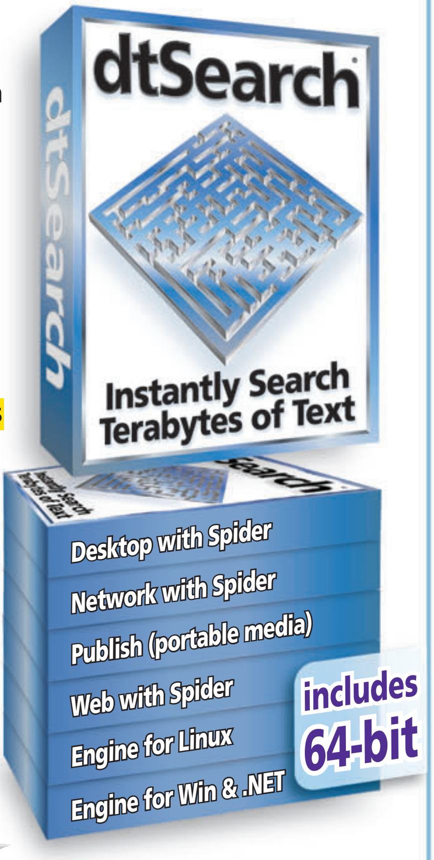
Silverlight

WPF

Business Intelligence

Instantly Search Terabytes of Text

- 25+ full-text and fielded data search options (with Unicode support for hundreds of international languages)
- Built-in file parsers / converters highlight hits in popular file types
- Spider supports static and dynamic web data; highlights hits with links, formatting and images intact
- API supports .NET, C++, Java, SQL, etc. .NET Spider API



The Smart Choice for Text Retrieval® since 1991

"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*

See www.dtsearch.com for hundreds more reviews, and hundreds of developer case studies

Fully-Functional Evaluations

1-800-IT-FINDS • www.dtsearch.com

Microsoft® msdn® magazine JANUARY 2010 VOLUME 25 NUMBER 1

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

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

CONTRIBUTING EDITORS Dino Esposito, Juval Lowy, Dr. James McCaffrey, John Papa, Charles Petzold, David Platt, Jeff Prosise, Jeffrey Richter, Aaron Skonnard, Stephen Toub

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
Christopher M. Coates Vice President, Finance & Administration
Abraham M. Langer Vice President, Digital Media, Audience Marketing
Erik A. Lindgren Vice President, Information Technology & Web 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 1081, Skokie, IL 60076-8081. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or Bleuchip International, P.O. Box 25542, London, ON N6C 6B2.

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.

BPA WORLDWIDE

Printed in the USA

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 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 2008

by Multi Edit Software

Multi-Edit 2008 delivers, a powerful IDE, with its speed, depth, and support for over 50 languages. Enhanced search functions include Perl 5 Regular Expressions and definable filters. Supports large DOS/Windows, UNIX, binary and Mac files. File Sync Integration for: Delphi 6, 7, 2005, C++ Builder 6, BDS 2006 and RadStudio 2007, VB 6, VC 6, VS 2003, 2005 and 2008. Includes file compare, code beautifying, command maps, and much more.

1-49 Users
Paradise #
A30 01201A01

$179.99

programmers.com/multiedit

VMware vSphere

Put time back into your day.

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

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

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

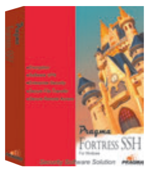
VMware Advanced Acceleration Kit for 6 processors
Paradise #
V55 78101A01

$9,234.99

programmers.com/vSphere

Pragma Fortress SSH—SSH Server & Client for Windows

by Pragma Systems



Certified for Windows 7/2008R2

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

Unify SQLBase Embedded Database

by Unify

NEW RELEASE!

SQLBase is an easy to deploy database for building desktop and Web applications in server environments without a DBA.

- Small footprint
- Zero admin deployment
- GUI admin
- Embed it and forget it

For applications with encrypted security, we offer SQLBase Treasury.

Reduce your database costs and simplify maintenance with Unify SQLBase.

Paradise #
C15 03101A01

$143.99

programmers.com/unify

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

FarPoint Spread for Windows Forms

NEW VERSION 5!

Now with Charting! The Best Grid is a Spreadsheet. Give your users the look, feel, and power of Microsoft® Excel®, without needing Excel installed on their machines. Join the professional developers around the world who consistently turn to FarPoint Spread to add powerful, extendable spreadsheet solutions to their COM, ASP.NET, .NET, BizTalk Server and SharePoint Server applications.

- World's #1 selling development spreadsheet
- Read/Write native Microsoft Excel Files
- Charts with 85 new chart types
- Fully extensible models
- Royalty-free, run-time free

Paradise #
F02 01101A01

$936.99

programmers.com/grapecity

Enterprise Architect 7.5

Visualize, Document and Control Your Software Project

by Sparx Systems

Enterprise Architect is a comprehensive, integrated UML 2.1 modeling suite providing key benefits at each stage of system development. Enterprise Architect 7.5 supports UML, SysML, BPMN and other open standards to analyze, design, test and construct reliable, well understood systems. Additional plug-ins are also available for Zachman Framework, MODAF, DoDAF and TOGAF, and to integrate with Eclipse and Visual Studio 2005/2008.

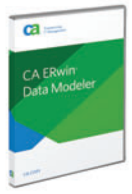
Corporate Edition
1-4 Users
Paradise #
SP6 03101A02

$182.99

programmers.com/sparxsystems

CA ERwin r7.3

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



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

$3,951.99

programmers.com/ca

FREE ON DEMAND WEBINAR SERIES: MORE Maximum Data Modeling with CA ERwin 7.3

In our last webinar series, we looked at CA ERwin's core functionality. In this second series, we provide a grounding in how CA ERwin r7.3's new features help you with Master Data Management, Metadata Management, Data Warehousing, Data Governance and Business Intelligence.

There are six sessions in the on demand series:

- What's New in CA ERwin 7.3
- MDM (Master Data Management) with CA ERwin and Data Profiling tool
- Collaborative model management with CA ERwin ModelManager
- Validate the integrity of your model with CA ERwin Validator
- Reporting: Crystal Reports, PDF, HTML
- SAPHIR Option: light at the end of the metadata tunnel

VIEW TODAY: programmers.com/MDM_2009

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™.

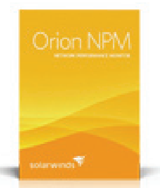
Professional Upgrade from any Active AS Pro + Silver Mtn
Paradise #
I21 09201S06

$4,228.99

programmers.com/flexera

Orion Network Performance Monitor

by Solarwinds



Orion Network Performance Monitor is a comprehensive fault and network performance management platform that scales with the rapid growth of your network and expands with your network management needs. It offers out-of-the-box network-centric views that are designed to deliver the critical information network engineers need. Orion NPM is the easiest product of its kind to use and maintain, meaning you will spend more time actually managing networks, not supporting Orion NPM.

Paradise #
S4A 08201E02

$4,606.99

programmers.com/solarwinds

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

Prices subject to change. Not responsible for typographical errors.

Two Guys in the Cloud

It's amazing what you can learn when you shut up and listen. Case in point: I was in a Super Shuttle van on the way to Los Angeles International airport in mid-November, having just finished my time at Microsoft PDC09. Sitting behind me were two guys talking about the show: one was a developer with Chevron, the other worked for a vendor that builds—among other things—plug-ins for Salesforce.com.

My natural inclination would be to ask them if they subscribe to *MSDN Magazine*, what they like and don't like about the magazine, stuff like that (by the way, if you'd like to chat with me about anything related to the magazine, please drop a line to me at kward@1105media.com). Instead, I held my tongue (no mean feat, if you know me) and just listened to them banter. I learned a lot.

For one thing, they were both intensely interested in the potential of the cloud, which is certainly music to Microsoft's ears. The Chevron guy was unsure of how much it could do for his company, but it sounds like he got a lot more information at the show. The vendor was much more well-versed in the cloud, of course, and seemed very knowledgeable in particular about Windows Azure.

Chevron Guy had some experience with both Amazon's cloud offering and Windows Azure. He said something very interesting. He liked both, but liked Windows Azure more—a good bit more. In fact, he said he'd probably drop his work with Amazon to move to Windows Azure. "Azure is a lot further along than I thought it was," he said.

Vendor Guy agreed. "I can confirm that," he said, or words very close to that effect. Chevron Guy mentioned how fast Windows Azure was. Vendor Guy said he'd seen the same thing. "Maybe it's because we're the only two people on it," he joked. They were both impressed with Windows Azure, even though Chevron Guy thought cloud computing would really pick up when it could be more useful. He sees it more as a solution in search of a problem.

That's where this issue, with its focus on Windows Azure, comes in. We've got some advice for both Vendor Guy and Chevron Guy in these pages.

For me, the cloud has no real future if it can't be secured. Moving across domain and other security boundaries is scary, and creates significant challenges for developers. Fortunately, Windows Azure was built with security in mind. As Jonathan Wiggs writes in his article about securing data in Windows Azure, "proper understanding of both encryption and the .NET security model will be needed by product designers and developers building on the Windows Azure platform." He dives deep into Windows Azure cryptography, key storage and security threats, and more.

Another key to the success of the cloud is being able to easily port existing applications and architectures to take advantage of its strengths. You don't want to be rewriting tons of code simply to move to the cloud—if that's the case, you probably won't move at all. Windows Azure also helps with this process, as we demonstrate in "Designing Services for Windows Azure." The authors take us through a fictional scenario where they do just that, involving a bank that moves its services into the cloud.

There's a lot more cloud goodness in this issue as well, and more coming in future issues.

So, Chevron and Vendor Guys, I hope we've helped sort out some Windows Azure and cloud issues for you. Thanks for talking.

Keith Ward

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: kward@1105media.com.

"Best Defect Tracker & Project Management Software"

-asp.netPRO Magazine

FREE! 1-User License

4 years in a Row!

READERS' CHOICE 2006 asp.net PRO WINNER
READERS' CHOICE 2007 asp.net PRO WINNER
READERS' CHOICE 2008 asp.net PRO WINNER
READERS' CHOICE 2009 asp.net PRO WINNER

OnTime 2009

- Defect Tracking
- Requirement Mgmt
- Help Desk
- Team Wiki
- Scrum/Agile

Used by Over 7,000 Dev Teams Worldwide

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

Team Wiki
Finally, one place to collaborate on projects in a clean, organized, hierarchical structure.

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

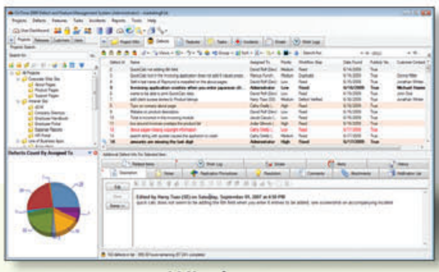
Scrum
Harness the power of scrum within the OnTime client. Use Sprints, Burndowns, Trending Charts and more.

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

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

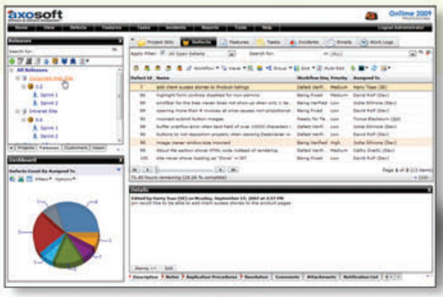
Customize
Unlimited custom fields, customize all field names, and decide when fields are visible, editable or required.

Security
Secure every aspect of the system by project, user roles, and even fields.


Windows

800.653.0024
Our U.S. and International sales teams look forward to answering your questions or setting up a live demo.
8:00 a.m. - 5:00 p.m. MST

Starts at: FREE
Single-user licenses of OnTime are always free. 5-user teams start at just $395. Visit axosoft.com for a specific quote.


Web

axosoft

Software for Software Development™

Visit **www.axosoft.com** for a video tour, live demos, downloads, and much more.

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®

UI Mockups, User Experience Tips, JavaScript Checker and More

Quickly Create UI Mockups

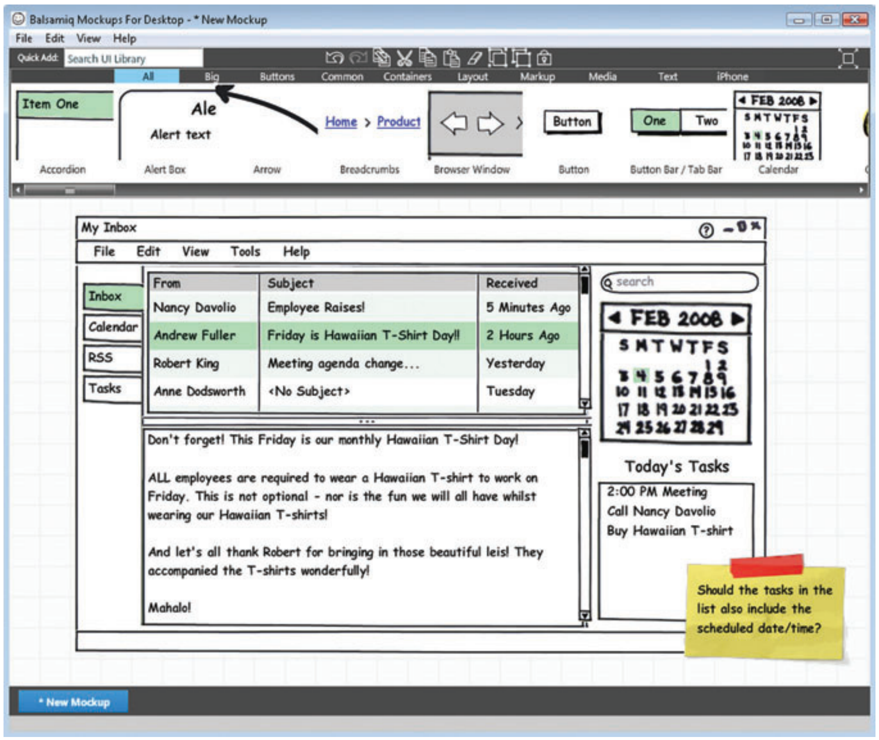
User interface mockups are an important tool in software engineering. During the requirements phase they serve as prototypes that elicit feedback from stakeholders and end users, and they're an integral part of any functional specification.

But what's the best way to create UI mockups? The simplest approach is to use pen and paper. It's quick and easy, requires no special software, and can be done by anyone, regardless of their technical savvy. Of course, such prototypes are harder to archive and are more difficult to share with remote stakeholders.

Alternatively, you can use programs like Microsoft Visio or Visual Studio Designer to build UI mockups that closely mimic what the finished product will look like. Unfortunately, creating such mockups takes considerably longer than using pen and paper. Also, showing non-technical stakeholders a "polished" mockup may lead them to believe that the work is nearly done.

Balsamiq Mockups For Desktop (version 1.6) is a UI mockup tool that combines the speed, simplicity, and low-tech look and feel of paper mockups with the archival and sharing benefits inherent in computer-generated mockups. And unlike Visual Studio and Visio, which are large and complex programs that serve many functions, Balsamiq Mockups has a focused goal: to make creating UI mockups as quick and painless as possible.

The Mockups For Desktop user interface is incredibly straightforward—there's a design surface and a list of common UI controls to add to your mockup, including buttons, labels, textboxes, checkboxes, tabs, combo boxes, hyperlinks, scroll bars, splitters and more. There are also pre-built controls for browser windows, video players and dialog boxes. In total, Balsamiq Mockups ships with more than 75 controls. You can also import any image file as a control or download user-created controls from the Balsamiq Studios LLC Web site.



From	Subject	Received
Nancy Davolio	Employee Raises!	5 Minutes Ago
Andrew Fuller	Friday is Hawaiian T-Shirt Day!!	2 Hours Ago
Robert King	Meeting agenda change...	Yesterday
Anne Dodsworth	<No Subject>	Tuesday

Don't forget! This Friday is our monthly Hawaiian T-Shirt Day!

ALL employees are required to wear a Hawaiian T-shirt to work on Friday. This is not optional - nor is the fun we will all have whilst wearing our Hawaiian T-shirts!

And let's all thank Robert for bringing in those beautiful leis! They accompanied the T-shirts wonderfully!

Mahalo!

Today's Tasks

2:00 PM Meeting
Call Nancy Davolio
Buy Hawaiian T-shirt

Should the tasks in the list also include the scheduled date/time?

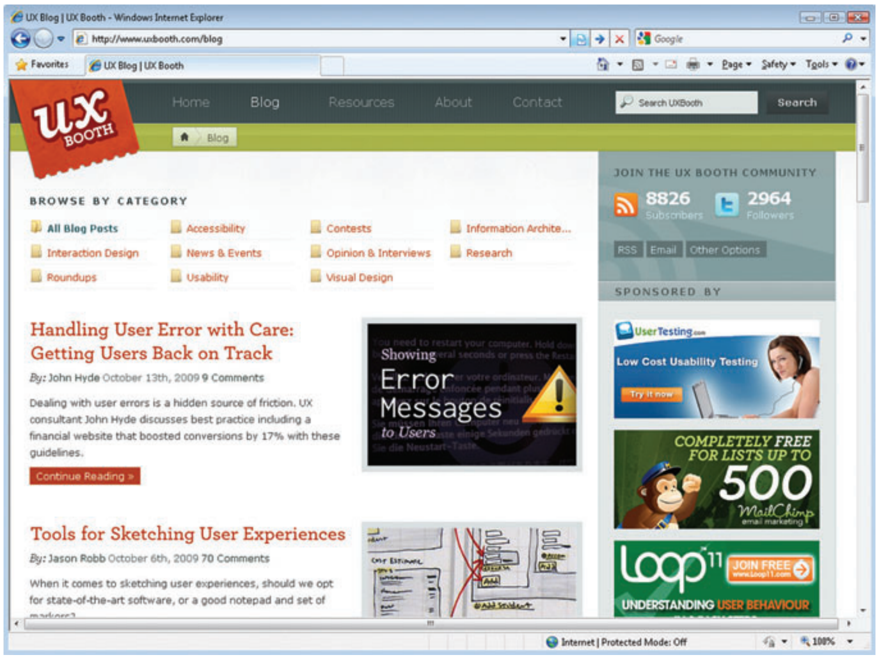
Figure 1 **A Quick Balsamiq UI Mockup**

Adding a control to your mockup is as simple as dragging it from the list of controls and dropping it onto the design surface. Once on the design surface, double-click the control to edit its contents. Hovering your mouse over a control displays a floating panel with an assortment of configuration options specific to that control. Every setting is right there in the floating panel—there are no menus or property windows you need to hunt and peck through. Having all of these settings right at your fingertips flattens the learning curve and greatly reduces the time it takes to create mockups.

Check out the mockup in **Figure 1**, which I created in three minutes, from start to finish. To mock the list of e-mails I dragged a Data Grid control onto the designer and then used my mouse to position and size it. Next, I double-clicked the grid, which displays its contents as comma-delimited text. I replaced the default contents with

All prices confirmed at press time are subject to change. The opinions expressed in this column are solely those of the author and do not necessarily reflect the opinions at Microsoft.

Send your questions and comments for Mitchell to toolsmm@microsoft.com.



Handling User Error with Care: Getting Users Back on Track

By: John Hyde October 13th, 2009 9 Comments

Dealing with user errors is a hidden source of friction. UX consultant John Hyde discusses best practice including a financial website that boosted conversions by 17% with these guidelines.

Continue Reading »

Showing Error Messages to Users

Tools for Sketching User Experiences

By: Jason Robb October 6th, 2009 70 Comments

When it comes to sketching user experiences, should we opt for state-of-the-art software, or a good notepad and set of markers?

UserTesting.com Low Cost Usability Testing Try it now

COMPLETELY FREE FOR LISTS UP TO 500 MailChimp email marketing

Loop11 JOIN FREE www.loop11.com UNDERSTANDING USER BEHAVIOUR

The UX Booth Blog

text to represent the three columns and four rows shown in **Figure 1**. I then hovered my mouse over the grid to bring up its properties in a floating window. From there I ticked a checkbox to add a scrollbar, used a slider to adjust the row height and chose to have the second row appear as selected.

Once a mockup has been created, it can be saved to disk, exported as a .PNG image or exported into an XML format. In addition to the regular version, a stripped-down version of Mockups For Desktop can be used for free from the company's Web site. And a collaborative, online version of the software is in the works.

Price: $79 per user license

balsamiq.com

Blogs of Note

The user experience (UX) is one of the most important aspects of a software application. Your users decide whether your application is a success or not, and they don't care about your application's architecture or that super-clever algorithm you spent a week perfecting. Unfortunately, most software developers—myself included—struggle with UX design. It's all too easy to get immersed in the low-level details of the application and leave UX design as an afterthought.

To stay reminded of the importance of UX design, and to pick up some great tips for improving your applications' user experiences, check out the **UX Booth Blog**, which includes submissions from several noted UX design authors, trainers and consultants. My favorite posts are those that offer specific tips for improving the UX of a common scenario. Building a contact form on your Web site? Be sure to read "Creating a Usable Contact Form," where author Matthew Kammerer shares advice on what information to display, how to lay it out on screen and what to do once the feedback has been submitted. And be sure to check out John Hyde's post, "Handling User Error with Care," in which he shares best practices on where to display error messages and how to word them.

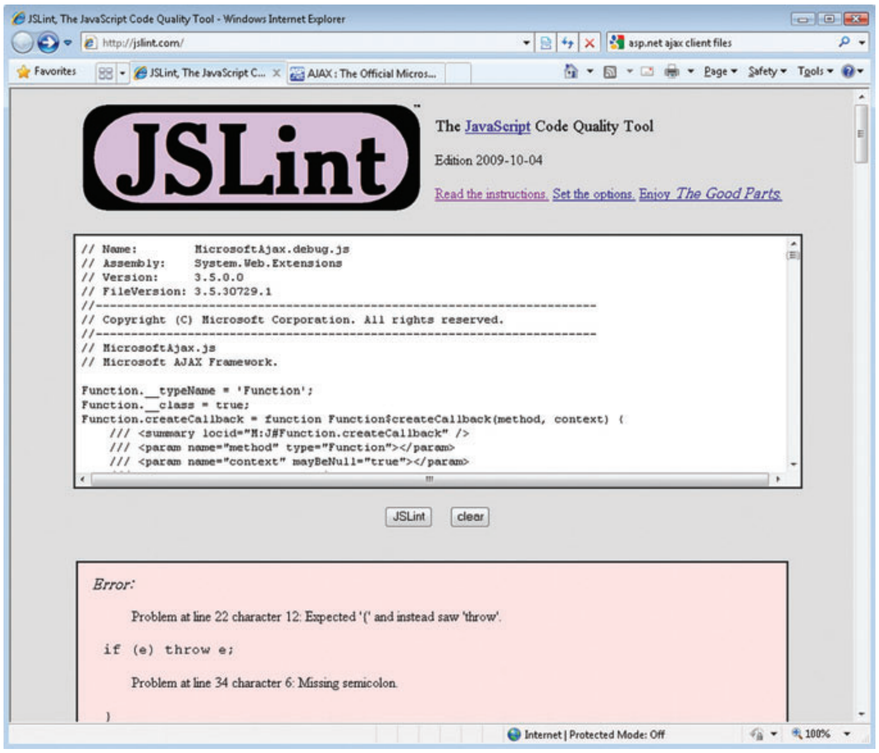
Other blog entries highlight how to improve the user experience through less direct means. One post describes how to improve the performance of a Web site by creating optimized images. Another entry shares tools for improving an application's accessibility. You'll also find more general posts, including book reviews, usability lessons learned from the trenches and interviews with usability experts.

uxbooth.com/blog

Check Your JavaScript

JavaScript is an interesting language. For much of its history, JavaScript was considered somewhat of a toy language, used mainly for performing simple tasks in Web pages. But JavaScript is a robust and powerful language, and today it's used to build rich, interactive Web applications.

Unfortunately, the JavaScript language has a number of design decisions that


JSLint, The JavaScript Code Quality Tool - Windows Internet Explorer
http://jslint.com/
asp.net ajax client files
Favorites
JSLint, The JavaScript C...
AJAX : The Official Micros...
JSLint
Read the instructions. Set the options. Enjoy The Good Parts
JSLint
clear
Internet | Protected Mode: Off
100%

The JavaScript Code Quality Tool

Edition 2009-10-04

```
// Name:        MicrosoftAjax.debug.js
// Assembly:    System.Web.Extensions
// Version:     3.5.0.0
// FileVersion: 3.5.30729.1
//-----------------------------------------------------------------------
// Copyright (C) Microsoft Corporation. All rights reserved.
//-----------------------------------------------------------------------
// MicrosoftAjax.js
// Microsoft AJAX Framework.

Function.__typeName = 'Function';
Function.__class = true;
Function.createCallback = function Function$createCallback(method, context) {
    /// <summary locid="M:J#Function.createCallback" />
    /// <param name="method" type="Function"></param>
    /// <param name="context" mayBeNull="true"></param>
```

Error:

```
Problem at line 22 character 12: Expected '(' and instead saw 'throw'.

if (e) throw e;

Problem at line 34 character 6: Missing semicolon.

}
```

Figure 2 **The JSLint JavaScript Code Quality Tool**

allow for poor programming practices and, if misused, can lead to bugs and less-maintainable code. For example, in C# every line must terminate with a semicolon. In JavaScript, most statements can end with either a semicolon or a carriage return, which can be confusing. Likewise, in C# every variable must be declared before it can be used. Not so in JavaScript.

JSLint is a free JavaScript Code Quality Tool created by **Douglas Crockford** that runs a variety of static analysis checks against a block of JavaScript code. By default, JSLint displays warnings when encountering global variables; statements not terminated with a semicolon; *if, while, do* and *for* statements that aren't followed by a statement block; and unreachable code, among other conditions. Additional checks are configurable through JSLint's options. For instance, you can instruct JSLint to disallow undefined variables, to disallow the unary increment and decrement operators ++ and --, and whether to allow the use of the *eval* function.

To use JSLint, visit JSLint.com, paste your JavaScript code into the textbox, select the options and click the "JSLint" button. JSLint will then parse your code and display a list of errors, as **Figure 2** shows. And because JSLint is written in JavaScript, it runs entirely on your browser, meaning that your code is not sent over the Internet. You can optionally download the JSLint source code from the Web site to run on your local environment, if you'd prefer.

Also check out **JSLint.VS**, a free Visual Studio Add-In created by Predrag Tomasevic that lets you run JSLint on a file or selected code block directly from the Visual Studio IDE. The errors identified by JSLint appear in the Task List window. You can even configure JSLint.VS to run on selected files or folders whenever the project is built.

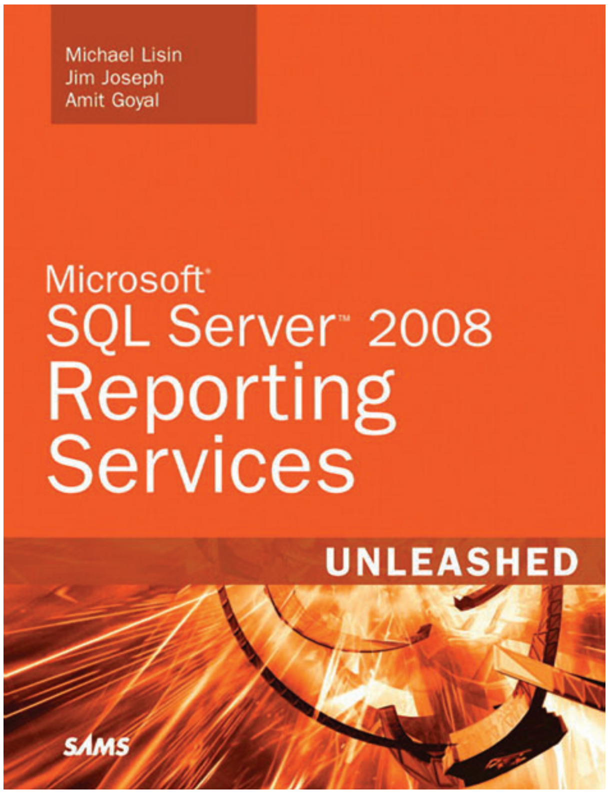
JSLint: jslint.com

JSLint.VS: jslint.codeplex.com

The Bookshelf

Over the past decade, storing structured information has become a trivial task. With modern databases and data-access frameworks, collecting data involves a sprinkle of drag and drop with just a dash of code. Coupled with decreasing storage costs and the increasing competitive advantage such information can bear, businesses are eager to catalog every possible data point.

Of course, such data is useless unless workers can access and assess the data in a meaningful way. Microsoft SQL Server Reporting Services (SSRS) is a server-based, enterprise-grade reporting platform that enables workers to create, explore and view reports.



Microsoft SQL Server 2008 Reporting Services Unleashed

Like with any enterprise-grade platform, SSRS is expansive in its features and use cases. I recently helped a client evaluate and get started with SSRS and found "Microsoft SQL Server 2008 Reporting Services Unleashed" (Sams, 2009) to be an invaluable guide for learning the ins and outs.

The book is divided into five parts. The first part provides a light overview of SSRS, highlights common user scenarios, introduces the SSRS architecture, and compares and contrasts different report deployment scenarios. There's also a short chapter on installing SSRS, with step-by-step instructions and plenty of screenshots.

The primary purpose of SSRS is to present data through reports and to allow workers to build, analyze and consume these reports. The book's second part explores how to author reports and examines topics like expressions, parameters, formatting, navigation, aggregation and ad hoc reports in-depth. During my project, I routinely bumped into roadblocks when designing reports. I'd get stumped trying to format data a certain way, or get stuck when needing certain information—like the date and time the report was generated—to show on the report. Most of these obstacles were side-stepped by using the book's index and thumbing through the chapters in this section.

Following report authoring, the book looks at managing SSRS. SSRS is a server-based technology; the reports (and data) reside on servers and can be accessed by clients in a variety of ways. Reports can be generated on-demand or on schedule, and clients can subscribe to reports and have them delivered through a file share or e-mail. These various options, along with other administration tasks, are covered here.

The book's last two sections look at how to customize and extend SSRS, along with ways to integrate SSRS and SharePoint.

"Microsoft SQL Server 2008 Reporting Services Unleashed" is an excellent introduction to SSRS for administrators, DBAs and users. The depth of material is a little light in some areas, but this book does an excellent job conveying the most important aspects and exploring the breadth of features and functionality available in SSRS.

Price: $49.99
informit.com/sams

SCOTT MITCHELL, *author of numerous books and founder of 4GuysFromRolla.com, is an MVP who has been working with Microsoft Web technologies since 1998. Mitchell is an independent consultant, trainer and writer. Reach him at Mitchell@4guysfromrolla.com or via his blog at ScottOnWriting.net.*

Get ready for…
a data layer you can build in **60 *seconds***

Presenting Telerik OpenAccess – finally an ORM that's easy to use.

Forward and Reverse Mapping Capabilities
OpenAccess meets your specific needs by letting you choose whether to start the mapping from an existing database or just persist the application classes that you already have. OpenAccess also supports a rich variety of mappings for collections and class hierarchies.

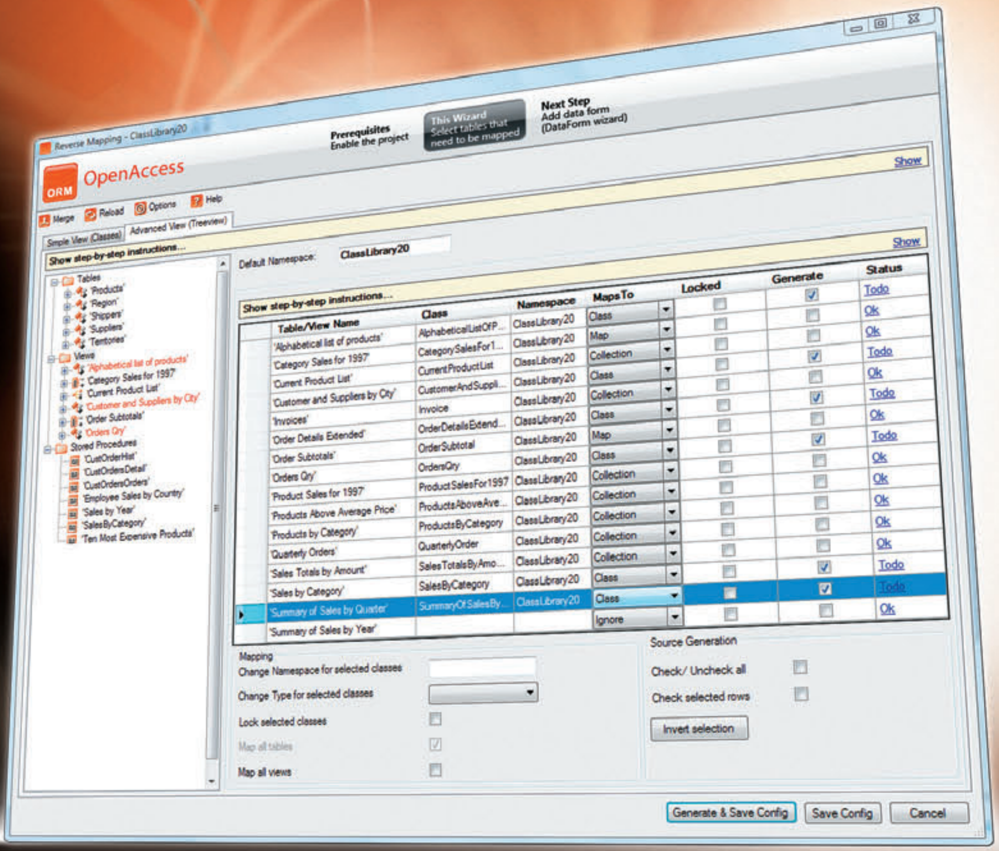
Tight Visual Studio Integration
OpenAccess it tightly integrated in Visual Studio .Net so you don't have to leave your favorite IDE. You will see your persistent classes in the solution explorer, nicely separated from your business logic code.

Intuitive Point-and-click Wizards
With OpenAccess ORM you can set up your persistent model codelessly and with just a few clicks. Intuitive yet powerful wizards will help you master the mapping process even if you haven't ever used an ORM before.

Full LINQ Support
OpenAccess ORM offers a complete support for LINQ, providing all features being used in the "Microsoft 101 LINQ Samples".

Advanced Features Out-of-the-box
OpenAccess offers an array of built-in heavy-duty features such as full support for SQL Azure, distributed level 2 cache, fetch plans, support for n-tier and disconnected applications, and many more.

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


Reverse Mapping - ClassLibrary20

Prerequisites
Enable the project

This Wizard
Select tables that need to be mapped

Next Step
Add data form
(DataForm wizard)

OpenAccess ORM

Merge | Reload | Options | Help

Simple View (Classes) | Advanced View (Treeview)

Show step-by-step instructions... Show

- Tables
  - 'Products'
  - 'Region'
  - 'Shippers'
  - 'Suppliers'
  - 'Territories'
- Views
  - 'Alphabetical list of products'
  - 'Category Sales for 1997'
  - 'Current Product List'
  - 'Customer and Suppliers by City'
  - 'Order Subtotals'
  - 'Orders Qry'
- Stored Procedures
  - 'CustOrderHist'
  - 'CustOrdersDetail'
  - 'CustOrdersOrders'
  - 'Employee Sales by Country'
  - 'Sales by Year'
  - 'SalesByCategory'
  - 'Ten Most Expensive Products'

Default Namespace: ClassLibrary20

Show step-by-step instructions... Show

Table/View Name	Class	Namespace	Maps To	Locked	Generate	Status
'Alphabetical list of products'	AlphabeticalListOfP...	ClassLibrary20	Class	☐	✓	Todo
'Category Sales for 1997'	CategorySalesFor1...	ClassLibrary20	Map	☐	☐	Ok
'Current Product List'	CurrentProductList	ClassLibrary20	Collection	☐	☐	Ok
'Customer and Suppliers by City'	CustomerAndSuppli...	ClassLibrary20	Class	☐	✓	Todo
'Invoices'	Invoice	ClassLibrary20	Collection	☐	☐	Ok
'Order Details Extended'	OrderDetailsExtend...	ClassLibrary20	Class	☐	✓	Todo
'Order Subtotals'	OrderSubtotal	ClassLibrary20	Map	☐	☐	Ok
'Orders Qry'	OrdersQry	ClassLibrary20	Class	☐	✓	Todo
'Product Sales for 1997'	ProductSalesFor1997	ClassLibrary20	Collection	☐	☐	Ok
'Products Above Average Price'	ProductsAboveAve...	ClassLibrary20	Collection	☐	☐	Ok
'Products by Category'	ProductsByCategory	ClassLibrary20	Collection	☐	☐	Ok
'Quarterly Orders'	QuarterlyOrder	ClassLibrary20	Collection	☐	☐	Ok
'Sales Totals by Amount'	SalesTotalsByAmo...	ClassLibrary20	Collection	☐	☐	Ok
'Sales by Category'	SalesByCategory	ClassLibrary20	Class	☐	✓	Todo
'Summary of Sales by Quarter'	SummaryOfSalesBy...	ClassLibrary20	Class	☐	✓	Todo
'Summary of Sales by Year'			Ignore	☐	☐	Ok

Mapping
Change Namespace for selected classes
Change Type for selected classes
Lock selected classes ☐
Map all tables ✓
Map all views ☐

Source Generation
Check/ Uncheck all ☐
Check selected rows ☐
Invert selection

Generate & Save Config | Save Config | Cancel

UI COMPONENTS	DATA	PRODUCTIVITY TOOLS	AUTOMATED TESTING	TFS TOOLS	CMS
ASP.NET AJAX	OpenAccess ORM	JustCode	Web Testing Tools	Work Item Manager	Sitefinity
Silverlight	Reporting			Project Dashboard	
ASP.NET MVC					
WinForms					
WPF					

www.telerik.com/ORM

telerik
deliver more than expected

DINO ESPOSITO

Master-Detail Views with the ASP.NET Ajax Library

In the past few columns, I explored a number of data-related features of the upcoming ASP.NET Ajax Library beta, which is now part of the CodePlex Foundation (CodePlex.org). The journey began last September with a look at some of the new features in Web Forms (msdn.microsoft.com/magazine/ee431529) and continued with some other stops in the territory of ASP.NET AJAX. In particular, I touched on client-side templates and data binding (msdn.microsoft.com/magazine/ee309508), conditional rendering (msdn.microsoft.com/magazine/ee335716) and live binding (msdn.microsoft.com/magazine/ee819084).

When you think of data-driven Web pages, most of the time what you really have in mind is a master-detail view of some cross-related data. Master-detail views are ideal for rendering one-to-many relationships, and such relationships are so common in the real world that a Web platform that doesn't provide an effective set of tools for that functionality is inadequate.

The DataView client control is the fundamental tool for building master-detail views in ASP.NET AJAX.

ASP.NET Web Forms has always provided strong support for data binding and a powerful set of data-source and data-bound server controls. In Web Forms, server controls do a great job of rendering hierarchies of data using nearly any possible combination of grids, lists, and drop-down boxes and supporting multiple levels of nesting.

The drawback of the views you get out of Web Forms server controls is not the effectiveness of the rendering, but the static condition.

Users who navigate within a master-detail view typically switch among master records and drill down into the details of the records that are of interest. This interaction is the essence of a master-detail view.

In a classic Web Forms scenario, each drill-down operation may trigger a postback. Many postbacks—and subsequent page reloads—are not what makes users happy these days.

An alternative exists, but it's not free of issues either. It basically consists of preloading any possible data the user might want to see. The data is then downloaded with the standard page and kept hidden using CSS styles. At the same time, any handler of user actions is rewritten to unveil hidden content rather than triggering a postback. As you can see, this is not an easy way to go.

The ASP.NET Ajax Library, in collaboration with jQuery, offers a much more powerful toolset and makes it possible to write smooth and effective master-detail views that post back asynchronously and only when strictly needed.

The Hidden Power of the DataView Control

The DataView client control is the fundamental tool for building master-detail views in the ASP.NET Ajax Library. Combined with the sys-attach feature of the ASP.NET Ajax Library and live binding, the control offers an unprecedented level of flexibility as far as functionality and layout are concerned. In particular, the DataView control can serve to generate both the master and detail views.

To arrange a master-detail view with the ASP.NET Ajax Library, you need to follow three basic steps. First, create the markup for the master and detail views. Second, attach an instance of the DataView control to each view as a behavior. Finally, use live binding (or just plain data-binding expressions) to populate with fresh data the visual layout of the detail view. Note that all the templates, binding and component creation can be done both declaratively and imperatively in code. Let's start with a simple example that serves the purpose of making you familiar with the approach and the tools available.

Building a Plain Master-Detail View

Here's a simple layout for the master-detail view. It basically consists of two DIV tags. The master DIV contains an unordered list of items; the detail DIV, instead, contains a child table:

```
<div id="masterView">
  <ul class="sys-template">
  ...
  </ul>
</div>
<div id="detailView">
  <table>
    <tr>
      <td> ... </td>
      <td> ... </td>
    </tr>
    ...
  </table>
</div>
```

Disclaimer: This article is based on a pre-release version of ASP.NET Ajax Library. All information is subject to change.

Send your questions and comments for Esposito to cutting@microsoft.com.

Code download available at code.msdn.microsoft.com/mag201001Ajax4.

LEADTOOLS®

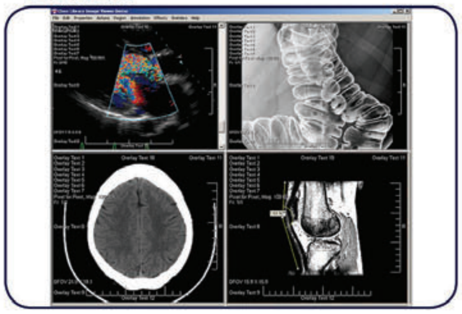
The World Leader in Imaging Development SDKs

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

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

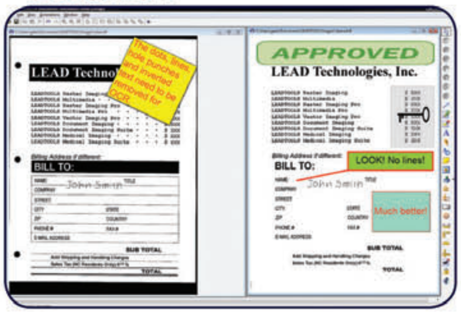
.NET, WPF, WCF, WF, C API, C++ Class Lib, COM & more!

Medical

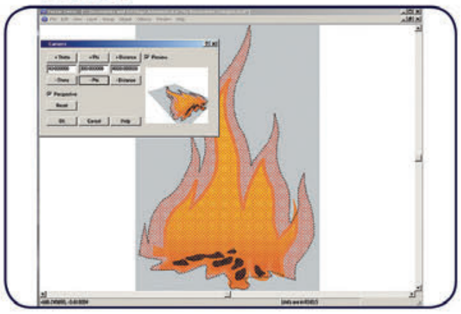


Multimedia

Document


LEAD Technologies, Inc.
APPROVED
The dots, lines, hole punches and inverted text need to be removed for OCR.
LOOK! No lines!
Much better!
BILL TO:

Vector



- **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.

Version 16.5 is a free upgrade for qualifying customers. To see if you qualify, or for more information on what is new in version 16.5, contact: **sales@LEADTOOLS.com**

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

LEAD TECHNOLOGIES INCORPORATED

High Level Design • Low Level Control

The LEADer in Intelligent Imaging

Figure 1 **The Master View**

```
<div>
  <ul class="sys-template" sys:attach="dataview"
    id="masterView"
    dataview:autofetch="true"
    dataview:dataprovider="/ajax40/mydataservice.asmx"
    dataview:fetchoperation="LookupCustomers"
    dataview:fetchparameters="{{ {query: 'A'} }}"
    dataview:selecteditemclass="selecteditem"
    dataview:initialselectedindex="0">
    <li>
      <span sys:command="Select">
      <span>{binding CompanyName}</span>
      ,&nbsp;&nbsp;
      <span>{binding Country}</span>
      </span>
    </li>
  </ul>
</div>
```

More often than not, the data to show in an AJAX page is retrieved from the Web server using a Web service, a Windows Communication Foundation (WCF) service and, of course, any services that can return JavaScript Object Notation (JSON). The data is commonly sent over the wire as a JSON stream. You can choose to manage the request to the data source yourself and use your own AJAX framework of choice such as Microsoft AJAX, jQuery or raw XmlHttpRequest calls.

The DataView control, however, also offers a sort of all-inclusive service. You point it to a remote data provider such as a Web service, indicate the operation to invoke and list the parameters. Any fetched data is automatically ready for display anywhere in the HTML page where the DataView is attached. **Figure 1** shows the markup code that's necessary for the master view.

The sys:attach attribute attaches a new instance of the DataView control to the UL tag. The code snippet doesn't show that, but it is necessary that you declare the "dataview" name and associate it to the JavaScript object that represents the behavior to attach. Typically, you declare the JavaScript objects you intend to use in the BODY tag.

```
<body xmlns:sys="javascript:Sys"
      xmlns:dataview="javascript:Sys.UI.DataView">
  ...
</body>
```

Figure 2 **The Detail View**

```
<div class="sys-template" sys:attach="dataview"
     dataview:data="{binding selectedData, source=$masterView}">
  <table>
    <tr>
       <td><b>Contact</b></td>
       <td><input id="contact" type="text"
                  sys:value="{{ContactName}}"/></td>
    </tr>
    <tr>
       <td><b>Address</b></td>
       <td><input id="address" type="text"
                  sys:value="{binding Street}"/></td>
    </tr>
    <tr>
       <td><b>City</b></td>
       <td><input id="city" type="text"
                  sys:value="{binding City}"/></td>
    </tr>
    <tr>
       <td><b>Phone</b></td>
       <td><input id="phone" type="text"
                  sys:value="{binding Phone}"/></td>
    </tr>
  </table>
</div>
```

Properties you set declaratively on the automatically created instance of the DataView define the remote call that will fetch data. In the example, I call the method LookupCustomers on MyDataService.asmx, passing a string parameter with the value of A.

In addition, the DataView control can accept a few properties specific to the master-detail scenario. The selectedItemClass property indicates the CSS style to be used for the elements in the item template that's currently marked as selected. The initialSelectedIndex property refers to the item in the view that must be marked as selected when the DataView first renders out its data.

The body of the UL tag contains the item template and it binds to fetched data via the ASP.NET Ajax Library live-binding syntax:

```
<span>{binding CompanyName}</span>
```

You could actually use a simpler data-binding expression here:

```
<span>{{ CompanyName }}</span>
```

A simple binding expression is enough if you have data to display that's read-only. If your code modifies displayed data and you need to see changes in real time, then you should opt for live binding. **Figure 2** shows the detail view.

There are many ways to generate a bunch of similar DOM elements on the fly.

You may have noticed that the value attribute is namespaced. Starting with ASP.NET Ajax Library beta, all attributes that contain {{expression}} or {binding ...} must include the Sys prefix to not be ignored.

The most interesting part of the code in **Figure 2** is the expression assigned to the data property of the DataView:

```
{binding selectedData, source=$masterView}
```

The syntax indicates that the values for the elements in the view will come from the object named masterView. The property that physically provides data to the detail view is selectedData. Needless to say, the object named masterView will have a property named selectedData, otherwise an error occurs. **Figure 3** shows the sample page in action.

More Control over the Fetch Process

In the first example, I configured the master DataView to support auto-fetching. This means the DataView object is responsible for triggering the specified fetch operation right after initialization.

This is definitely a good choice for many applications, but some scenarios require more control over the fetch process. In particular, it's often required that you start the fetch following a user action. The next example will rework the previous code to add a button bar where you choose the first initial of the name of the customers you want to see listed. **Figure 4** shows the final screen.

There are many ways to generate a bunch of similar DOM elements on the fly. You can go with the raw DOM API or perhaps resort to the more abstract programming interface of the Microsoft AJAX library. Or you can opt for jQuery. Here's a code snippet using the services of the jQuery library to generate a button for each possible initial of the customer name:

```
for(var i=0; i<26; i++) {
  var btn = $('<input type="button"
    onclick="filterQuery(this)" />');
  var text = String.fromCharCode('A'.charCodeAt(0) + i);
  btn.attr("value", text).appendTo("#menuBar").show();
}
```

The $ function returns a DOM element resulting from the specified markup string. You then set its value property to an uppercase letter of the alphabet and append the DOM object to a placeholder down the page. The code runs from within the pageLoad function.

Each input button added in this way is bound to the same click handler. The click handler takes a reference to the DOM object and updates the master view. Here's the source code of the click handler:

```
function filterQuery(button) {
  // Enables live binding on the internal object that contains
  // the current filter
  Sys.Observer.setValue(currentQuery, "Selection", button.value);

  // Update the master view
  fillMasterView(currentQuery);
}
```

Note that, in this example, you need to track the current filter applied to select only a subset of customers. To avoid pain with the binding, and also to leave room for future enhancement, I opted for a custom object with a single property. Global to the page, the object is initialized as follows:

```
var currentQuery = { Selection: "A" };
```

The current selection is also displayed through a data-bound label in the page. Note the use of the namespace with the innerHTML attribute of the SPAN tag:

```
<h3>
  Selected customers:
  <span sys:innerhtml=
    "{binding Selection, source={{currentQuery}}}">
  </span>
</h3>
```

Next, when the user clicks a button to change the selection, you update the Selection property of the object. Note that the most straightforward code shown here won't really work:

```
currentQuery.Selection = button.value;
```

You must enter observable changes only via the setValue method. This is demonstrated in the code snippet shown earlier using the Sys.Observer.setValue method.

What about the code that fills up the master view programmatically?

To start off, you need to get hold of the instance of the DataView control that operates behind the master view. As mentioned, the sys-key attribute is only used internally for data-binding purposes. To retrieve a component like DataView you need to access the list of registered application components as exposed by the $find method of the Microsoft AJAX library. You use the ID of the root tag as a selector:

```
var dataViewInstance = Sys.get("$masterView");
```

In my example, the masterDataView is intended to be the ID of the UL tag marked with the sys-template attribute that renders the master view:

```
<div>
  <ul class="sys-template"
    sys:attach="dataview"
    ID="masterDataView"
    ...>
  ...
  </ul>
</div>
```

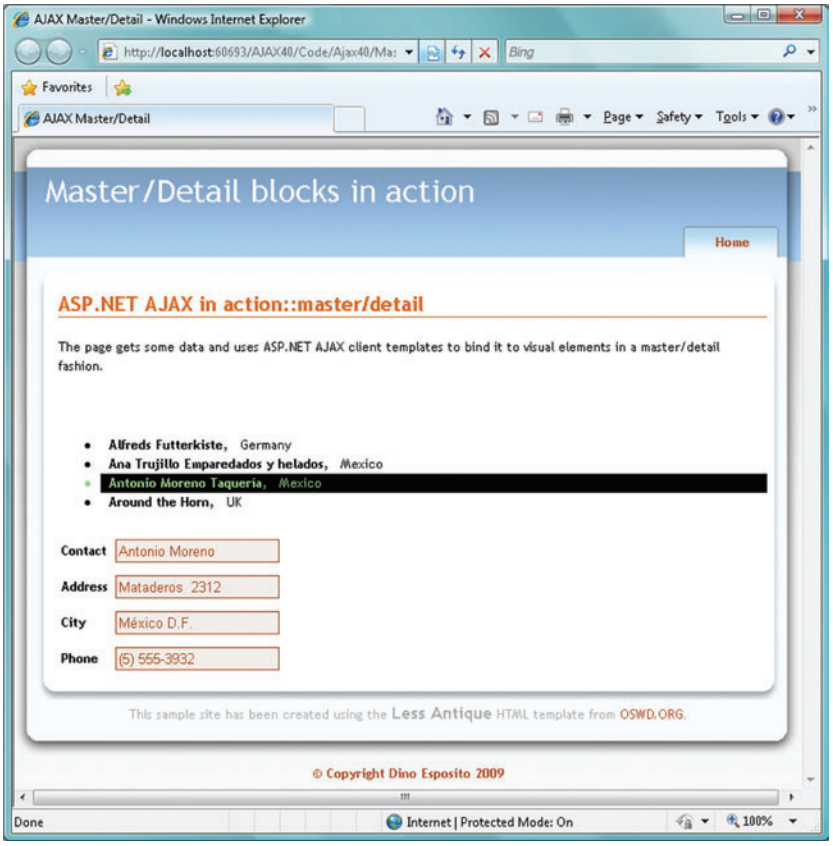
Once you have the DataView instance that populates the master view, adjust the fetch parameters and tell the DataView to get fresher data. Here's the code you need:

```
function fillMasterView(query) {
  // Retrieves the DataView object being used
  var dataViewInstance = Sys.get("$masterDataView");

  // DataView fetches fresh data to reflect current selection
  var filterString = query.Selection;
  dataViewInstance.set_fetchParameters({ query: filterString });
  dataViewInstance.fetchData();
}
```

The fetchData method on the DataView control uses the currently set provider, operation and parameters to place a remote call and refresh the view with downloaded data.

Adding Caching Capabilities

Let's consider the actual behavior of the page shown in **Figure 4**. A remote (and asynchronous) request is placed every time you click on a button to select a subset of customers. Subsequent selections to see details of a particular customer don't


AJAX Master/Detail - Windows Internet Explorer
http://localhost:60693/AJAX40/Code/Ajax40/Ma:
Bing
Favorites
AJAX Master/Detail
Page
Safety
Tools
Home
Done
Internet | Protected Mode: On
100%

Master/Detail blocks in action

ASP.NET AJAX in action::master/detail

The page gets some data and uses ASP.NET AJAX client templates to bind it to visual elements in a master/detail fashion.

- Alfreds Futterkiste, Germany
- Ana Trujillo Emparedados y helados, Mexico
- Antonio Moreno Taquería, Mexico
- Around the Horn, UK

Contact	Antonio Moreno
Address	Mataderos 2312
City	México D.F.
Phone	(5) 555-3932

This sample site has been created using the Less Antique HTML template from OSWD.ORG.

© Copyright Dino Esposito 2009

Figure 3 **A Master-Detail View Based on the DataView Control**

require a roundtrip as long as the data to display is already available for binding.

That mostly depends on what the fetch operation really returns. In my example, the LookupCustomers method is designed as follows:

```
public IList<Customer> LookupCustomers(string query);
```

The properties of the Customer class form the set of data you have ready for binding at no extra cost. If you want to display, say, the list of orders for each customer, then you can do that without placing an additional request, but only if the orders are packed with the Customer class and are sent over the wire with the first request.

ASP.NET Web Forms has always provided strong support for data binding and a powerful set of data source and data-bound server controls.

In a future article, I plan to tackle lazy-loading scenarios and mix that with the AJAX-specific Predictive Fetch pattern. For now, though, let's simply assume that the data you get out of the DataView fetch operation is enough for you to craft an effective user interface.

With the solution arranged so far, if you request customers whose name begins with "A" twice or more, then distinct requests are placed to get you basically the same set of data. A possible way to improve this aspect is adding some client-side caching capabilities. The jQuery library comes in handy as it provides an excellent local, in-memory cache exposed via its core functions.

As one alternative, you can make it so the response is cached by the browser. Then, even though you're issuing another XmlHttpRequest for the data, the browser doesn't really make a new request for it.

The jQuery Cache API

Seen from the outside, the jQuery cache is nothing more than an initially empty array that gets populated with keys and values. You can work with the jQuery cache API at two levels: The low-level API is represented by the cache array property; the higher-level data function provides a bit more abstraction, which saves you from having to check the array against nullness.

Here's the code to create an entry in the cache and store some data into it:

```
// Initializes the named cache if it doesn't exist
if (!jQuery.cache["YourNamedCache"])
   jQuery.cache["YourNamedCache"] = {};

// Stores a key/value pair into the named cache
jQuery.cache["YourNamedCache"][key] = value;
```

More precisely, the jQuery cache is organized as a set of named caches grouped under the global cache array. Reading a value from the cache requires the following code:

```
var cachedInfo;
if (jQuery.cache["YourNamedCache"])
  cachedInfo = jQuery.cache["YourNamedCache"][key];
```

In this way, you can gain total control over the organization of data within the cache. Each named cache is created on a strict on-demand basis and no duplication of data ever occurs.

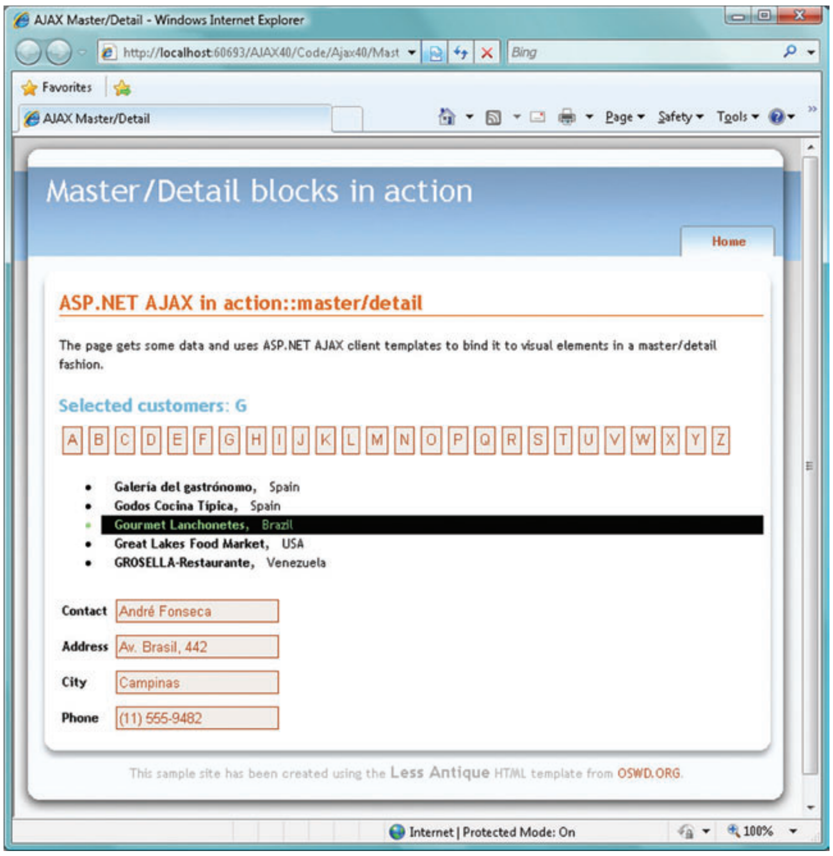
The data method offers a slightly richer programming interface that encapsulates some of the preliminary checks about the existence of a given named cache. Moreover, the data method allows you to attach the named cache to one or more DOM elements. The data method offers a basic get/put interface for you to read and write data items to the cache.

Here's a sample command to assign a given value to a key created in the cache associated with a given DOM element:

```
$('#elemID').data(key, value);
```

The named cache is created on-demand when the code attempts to access or manipulate the content. The library creates a named cache for the specified element and decides about its name. The name is a progressive number stored as an expando attribute to the DOM element.

The data method works on the content of a wrapped set. If you define a wrapped set that returns multiple nodes, then each element gets its own named cache that contains the same data. However, no real data duplication occurs because the same content is referenced—not cloned—across the various cache items.


AJAX Master/Detail - Windows Internet Explorer
http://localhost:60693/AJAX40/Code/Ajax40/Mast
Bing
Favorites
AJAX Master/Detail
Page
Safety
Tools
Home
Internet | Protected Mode: On
100%

Master/Detail blocks in action

ASP.NET AJAX in action::master/detail

The page gets some data and uses ASP.NET AJAX client templates to bind it to visual elements in a master/detail fashion.

Selected customers: G

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- Galería del gastrónomo, Spain
- Godos Cocina Típica, Spain
- Gourmet Lanchonetes, Brazil
- Great Lakes Food Market, USA
- GROSELLA-Restaurante, Venezuela

Contact	André Fonseca
Address	Av. Brasil, 442
City	Campinas
Phone	(11) 555-9482

This sample site has been created using the Less Antique HTML template from OSWD.ORG.

Figure 4 **Starting Data Binding On-Demand**

Figure 5 **Caching Fetched Data**

```
var currentQuery = { Selection: "A" };

function pageLoad() {
  // Build the button bar to select customers by initial
  for(var i=0; i<26; i++) {
    var btn = $('<input type="button"
         onclick="filterQuery(this)" />');
    var text = String.fromCharCode('A'.charCodeAt(0) + i);
    btn.attr("value", text).appendTo("#menuBar").show();
  }

  // Refresh the list of customers
  fillMasterView(currentQuery);
}
function filterQuery(button) {
   Sys.Observer.setValue(currentQuery, "Selection", button.value);

   // Updates the master view
   fillMasterView(currentQuery);
}
function fillMasterView(query) {
   // Check cache first: if not, go through the data provider
   if (!reloadFromCache(query))
       reloadFromSource(query);
}
function reloadFromCache(query) {
   // Using the query string as the cache key
   var filterString = query.Selection;

   // Check the jQuery cache and update
   var cachedInfo = $('#viewOfCustomers').data(filterString);
   if (typeof (cachedInfo) !== 'undefined') {
       var dataViewInstance = Sys.get("$masterView");
       dataViewInstance.set_data(cachedInfo);
       // Template automatically refreshed
       return true;
   }
   return false;
}
function reloadFromSource(query) {
   // Set the query string for the provider
   var filterString = query.Selection;

   // Tell the DataView to fetch
   var dataViewInstance = Sys.get("$masterView");
   dataViewInstance.set_fetchParameters({ query: filterString });
   dataViewInstance.fetchData(
          cacheOnFetchCompletion, null, null, filterString);
}
function cacheOnFetchCompletion(fetchedData, filterString) {
    if (fetchedData !== null) {
       $('#viewOfCustomers').data(filterString, fetchedData);
    }
}
```

Consider the following example:

```
$('div').data('A', fetchedData);
```

The code attempts to store an entry with a key of "A" in a named cache for each DIV tag you happen to have in the page. In this way, you can retrieve or set data from any of the DIV tags you have in the page. For example, the following two lines of code retrieve the same data:

```
// Data is actually stored in one place but referenced from many
var data1 = $('div').data('A');

// If the ID references a DIV in the same page,
// the returned data is the same as with the previous code
var data2 = $('#ThisElementIsDiv').data('A');
```

A common way of using the jQuery cache API is storing data to the DOM element that's really using it. In my example, the canonical solution would be caching customers in the DIV element bound to the master view.

Putting It All Together

Figure 5 shows the final version of the sample code that retrieves data from a remote service, caches it locally using jQuery and displays it via a DataView.

The main refactoring regards the way in which the master view is filled. You first check whether the data is available in the local cache and proceed with a remote request if no data can be found.

The reloadFromCache method returns a Boolean value to signify that data has been successfully loaded from the cache. You use the DataView's set_data method to assign the control a new data source. Next, you call the method refresh to update the view.

You store data in the cache after you retrieve it from the specified provider. The point is that the fetchData method works asynchronously. This means you can't just place a call to the method get_data right after the method fetchData returns:

```
dataViewInstance.set_fetchParameters({ query: filterString });
dataViewInstance.fetchData();
// At this point the method get_data can't retrieve yet
// data for the new selection.
```

However, the method fetchData accepts some parameters on the command line and all you have to do is pass a callback, as shown here:

```
dataViewInstance.fetchData(
   cacheOnFetchCompletion,    // success callback
   null,                      // failure callback
   null,                      // merge option: append/overwrite
   filterString);             // context
```

The first argument indicates the success callback that will be asynchronously invoked after the fetch terminates. The second argument is the callback to be invoked in case of failure. The third argument refers to the merge option. It is AppendOnly by default, or it can be OverwriteChanges. Both values are only relevant if you hook up the DataView to an AdoNetDataContext object. Finally, the fourth argument is the container for any data you want to be received by the success callback.

Here's the signature of the success callback:

```
function onSucceededFetch(fetchedData, context)
```

The first argument the callback receives is the fetched data. The second argument is any context information you specified through the caller. In my example, the fetch completion callback is the ideal place where to cache fetched data. The context parameter will be just the query string to cache by.

Data binding is a delicate art and requires a lot of attention and resources to be effective in an AJAX scenario. ASP.NET Ajax Library offers a lot of tools for crafting a valid data binding and master-detail solution. The list includes observable collections, live-binding syntax and the DataView control. The ASP.NET Ajax Library beta can be downloaded from ajax.codeplex.com. ■

DINO ESPOSITO *is an architect at IDesign and the co-author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. You can join his blog at weblogs.asp.net/despos.*

THANKS to the following technical experts for reviewing this article: *Dave Reed and Boris Rivers-Moore*

Internal Domain Specific Languages

Domain Specific Languages (DSLs) have been a popular topic over the past couple years and will probably grow in importance in years to come. You might already be following the "Oslo" project (now called SQL Server Modeling) or experimenting with tools such as ANTLR to craft "external" DSLs. A more immediately approachable alternative is to create "internal" DSLs that are written within an existing programming language such as C#.

Internal DSLs may not be quite as expressive and readable to non-developers as external DSLs that can read like English, but the mechanics of creating an internal DSL are simpler because you are not employing compilers or parsers external to your code.

Please note that I am not suggesting that the DSLs in this article are suitable for review by business experts. For this article, I will focus only on how the patterns of internal DSLs can make our jobs as developers easier by crafting APIs that are easier to read and write.

The mechanics of creating an internal DSL are simpler because you are not employing compilers or parsers external to your code.

I am pulling a lot of examples out of two open source projects written in C# that I administer and develop. The first is StructureMap, one of the Inversion of Control (IoC) Container tools for the Microsoft .NET Framework. The second is StoryTeller, an acceptance-testing tool. You can download the complete source code for both projects via Subversion at https://structuremap.svn.sourceforge.net/svnroot/structuremap/trunk or storyteller.tigris.org/svn/storyteller/trunk (registration required). I can also suggest the Fluent NHibernate project (fluentnhibernate.org) as another source of examples.

Literal Extensions

One of the more important points I want to make in this article is that there are many small tricks you can do to make your code read more cleanly and be more expressive. These small tricks can really add value to your coding efforts by making code easier to write correctly as it becomes more declarative and more intention-revealing.

More and more frequently I use extension methods on basic objects such as strings and numbers to reduce repetitiveness in the core .NET Framework APIs and to increase readability. This pattern of extending value objects is called "literal extensions."

Let's start with a simplistic example. My current project involves configurable rules for reoccurring and scheduled events. We initially attempted to create a small internal DSL for configuring these events (we are in the process of moving to an external DSL approach instead). These rules depend heavily on TimeSpan values for how often an event should reoccur, when it should start and when it expires. That might look like this snippet:

```
x.Schedule(schedule =>
{
    // These two properties are TimeSpan objects
    schedule.RepeatEvery = new TimeSpan(2, 0, 0);
    schedule.ExpiresIn = new TimeSpan(100, 0, 0, 0);
});
```

In particular, pay attention to "new TimeSpan(2, 0, 0)" and "new TimeSpan(100, 0, 0, 0)." As an experienced .NET Framework developer you may parse those two pieces of code to mean "2 hours" and "100 days," but you had to think about it, didn't you? Instead, let's make the TimeSpan definition more readable:

```
x.Schedule(schedule =>
{
    // These two properties are TimeSpan objects
    schedule.RepeatEvery = 2.Hours();
    schedule.ExpiresIn = 100.Days();
});
```

All I did in the sample above was use some extension methods on the integer object that return TimeSpan objects:

```
public static class DateTimeExtensions
{
    public static TimeSpan Days(this int number)
    {
        return new TimeSpan(number, 0, 0, 0);
    }

    public static TimeSpan Seconds(this int number)
    {
        return new TimeSpan(0, 0, number);
    }
}
```

In terms of implementation, switching from "new TimeSpan(2, 0, 0, 0)" to "2.Days()" isn't that big of a change, but which one is easier to read? I know that when I'm translating business rules into code, I'd rather say two days than "a time span consisting of two days, zero hours and zero minutes." The more readable version of the code is easier to scan for correctness, and that's enough reason for me to use the literal expression version.

Semantic Model

When I build a new DSL I need to solve two problems. First, my team and I start by asking ourselves how we'd like to express the DSL in a logical and self-describing way that will make it easy to

Send your questions and comments for Miller to mmpatt@microsoft.com.

VeriSign®

SHRINKWRAP YOUR APP.

WITH PROVEN CODE SIGNING FROM VERISIGN.

You developed the software. Now deliver it with the same care and vigilance by using VeriSign® Code Signing. Why? Code signing not only protects the identity and reputation of the author, but it also verifies the authenticity and version of your software. Then VeriSign helps you go a step further. It can create a unique digital signature every time the code is signed, and it supports more certification programs and development platforms than any other Certificate Authority. And you can leverage the reputation of the most recognized and trusted name in online security—VeriSign.

Learn how Code Signing from VeriSign can help make sure your applications are more trusted and adopted at **www.VeriSign.com/CodeSigning** or call 1-866-893-6565.

TRUST THE

Figure 1 **DSL Is Implemented on the ScreenActionClass**

```
public class ScreenObjectRegistry : IScreenObjectRegistry
{
    private readonly List<ScreenAction> _actions =
       new List<ScreenAction>();
    private readonly IContainer _container;
    private readonly ArrayList _explorerObjects = new ArrayList();
    private readonly IApplicationShell _shell;
    private readonly Window _window;

    public IEnumerable<ScreenAction> Actions {
       get { return _actions; } }


    public IActionExpression Action(string name)
    {
        return new BindingExpression(name, this);
    }

    // Lots of other methods that are not shown here
}
```

use. As much as possible, I try to do this without regard for how the actual functionality will be structured or built.

For example, the StructureMap Inversion of Control (IoC) container tool allows users to configure the container explicitly inside StructureMap's "Registry DSL" like this:

```
var container = new Container(x =>
{
    x.For<ISendEmailService>().HttpContextScoped()
        .Use<SendEmailService>();
});
```

If you aren't already familiar with the usage of an IoC container, all that code is doing is stating that when you ask the container at runtime for an object of type ISendEmailService, you will get an instance of the concrete type SendEmailService. The call to HttpContextScoped directs StructureMap to "scope" the ISendEmailService objects to a single HttpRequest, meaning that if the code is running inside ASP.NET, there will be a single unique instance of ISendEmailService for each individual HTTP request no matter how many times you request an ISendEmailService within a single HTTP request.

Once I have an idea for the desired syntax, I'm left with the crucial question of exactly how I'm going to connect the DSL syntax to code that implements the actual behavior. You could place the behavioral code directly into the DSL code such that runtime actions happen directly in Expression Builder objects, but I would strongly recommend against this in any but the most simplistic cases. The Expression Builder classes can be somewhat difficult to unit test, and debugging by stepping through a fluent interface is not conducive to either productivity or your sanity. You really want to put yourself in a position to be able to unit test (preferably), debug and troubleshoot the runtime behavioral elements of your DSL without having to step through all the code indirection in a typical fluent interface.

I need to build the runtime behavior and I need to craft a DSL that expresses the DSL user's intent as cleanly as possible. In my experience, it has been extremely helpful to separate the runtime behavior into a "semantic model," defined by Martin Fowler as "The domain model that's populated by a DSL" (martinfowler.com/dslwip/SemanticModel.html).

The key point about the previous code snippet is that it doesn't do any real work. All that little bit of DSL code does is configure the semantic model of the IoC container. You *could* bypass the fluent interface above and build the semantic model objects yourself like this:

```
var graph = new PluginGraph();
PluginFamily family = graph.FindFamily(typeof(ISendEmailService));

family.SetScopeTo(new HttpContextLifecycle());
Instance defaultInstance = new SmartInstance<SendEmailService>();
family.AddInstance(defaultInstance);
family.DefaultInstanceKey = defaultInstance.Name;

var container = new Container(graph);
```

The Registry DSL code and the code directly above are identical in runtime behavior. All the DSL does is create the object graph of the PluginGraph, PluginFamily, Instance and HttpContextLifecycle objects. So the question is, why bother with two separate models?

First of all, as a user I definitely want the DSL version of the two previous code samples because it's far less code to write, more cleanly expresses my intent and doesn't require the user to know very much about the internals of StructureMap. As the implementer of StructureMap, I need an easy way to build and test functionality in small units, and that's relatively hard to do with a fluent interface by itself.

With the semantic model approach, I was able to build and unit test the behavioral classes quite easily. The DSL code itself becomes very simple because all it does is configure the semantic model.

This separation of DSL expression and semantic model has turned out to be very beneficial over time. You will frequently have to iterate somewhat with your DSL syntax to achieve more readable and writeable expressions based on feedback from usage. That iteration will go much more smoothly if you don't have to worry quite so much about breaking runtime functionality at the same time you're changing syntax.

On the other hand, by having the DSL as the official API for StructureMap, I've on several occasions been able to extend or restructure the internal semantic model without breaking the DSL syntax. This is just one more example of benefits of the "Separation of Concerns" principle in software design.

Fluent Interfaces and Expression Builders

A fluent interface is a style of API that uses method chaining to create a terse, readable syntax. I believe the most well-known example is probably the increasingly popular jQuery library for JavaScript development. jQuery users will quickly recognize code such as the following:

```
var link = $('<a></a>').attr("href", "#").appendTo(binDomElement);
$('<span></span>').html(binName).appendTo(link);
```

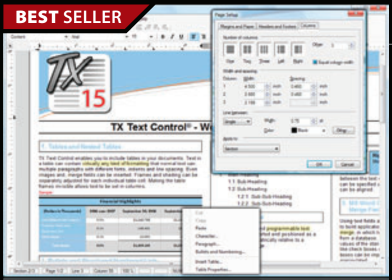
A fluent interface lets me "densify" code into a smaller window of text, potentially making the code easier to read. Also, it often helps me guide the user of my APIs to select the proper choices. The simplest and perhaps most common trick in making a fluent interface is to simply make an object return itself from method calls (this is largely how jQuery works).

I have a simple class I use in StoryTeller to generate HTML called "HtmlTag." I can build up an HtmlTag object quickly with method chaining like this:

```
var tag = new HtmlTag("div").Text("my text").AddClass("collapsible");
```

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 14,000 customers and 250,000 users in 110 countries

BEST SELLER

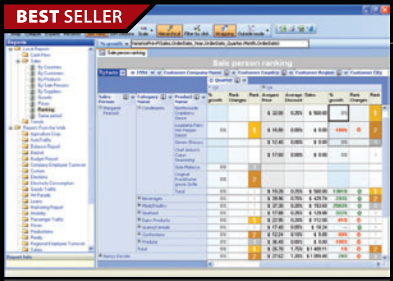
Janus WinForms Controls Suite | from $757.44

Janus systems

Add Outlook style interfaces to your WinForms applications.

- Janus GridEX for .NET (Outlook style grid)
- Janus Schedule for .NET and Timeline for .NET (Outlook style calendar view and journal)
- Janus ButtonBar for .NET and ExplorerBar for .NET (Outlook style shortcut bars)
- Janus UI Bars and Ribbon Control (menus, toolbars, panels, tab controls and Office 2007 ribbon)
- Now includes Office 2007 visual style for all controls

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 VISA DISCOVER

GSA Schedule Contract GS-35F-0188R

Internally, the HtmlTag object is just returning itself from the calls to Text and AddClass:

```
public HtmlTag AddClass(string className)
{
    if (!_cssClasses.Contains(className))
    {
        _cssClasses.Add(className);
    }

    return this;
}
public HtmlTag Text(string text)
{
    _innerText = text;
    return this;
}
```

In a more complicated scenario you may separate the fluent interface into two parts, the semantic model that supplies the runtime behavior (more on this pattern later) and a series of "Expression Builder" classes that implement the DSL grammars.

Users of this fluent interface should only specify the action name and the keyboard shortcut keys once. After that, the user shouldn't have to see those methods in IntelliSense.

I use an example of this pattern in the StoryTeller user interface for defining keyboard shortcuts and dynamic menus. I wanted a quick programmatic way to define a keyboard shortcut for an action in the user interface. Also, because most of us can't remember every keyboard shortcut for each application we use, I wanted to create a single menu in the UI that exposed all the available shortcuts and the keyboard combinations to run them. Also, as screens are activated in the main tab area of the StoryTeller UI, I wanted to add dynamic menu strip buttons to the UI that were specific to the active screen.

I certainly could have just coded this the idiomatic Windows Presentation Foundation (WPF) way, but this would have meant editing a couple different areas of XAML markup for keyboard gestures, commands, the menu strip objects for each screen and menu items–and then making sure that these were all correctly tied together. Instead, I wanted to make this registration of new shortcuts and menu items as declarative as possible, and I wanted to reduce the surface area of the code to a single point. I of course made a fluent interface that configured all the disparate WPF objects for me behind the scenes.

In usage, I can specify a global shortcut to open the "Execution Queue" screen with the following code:

```
// Open the "Execution Queue" screen with the
// CTRL - Q shortcut
Action("Open the Test Queue")
    .Bind(ModifierKeys.Control, Key.Q)
    .ToScreen<QueuePresenter>();
```

In the screen activation code for an individual screen, I can define temporary keyboard shortcuts and the dynamic menu options in the main application shell with code like this:

```
screenObjects.Action("Run").Bind(ModifierKeys.Control, Key.D1)
   .To(_presenter.RunCommand).Icon = Icon.Run;

screenObjects.Action("Cancel").Bind(ModifierKeys.Control, Key.D2)
   .To(_presenter.CancelCommand).Icon = Icon.Stop;

screenObjects.Action("Save").Bind(ModifierKeys.Control, Key.S)
   .To(_presenter.SaveCommand).Icon = Icon.Save;
```

Now, let's take a look at the implementation of this fluent interface. Underlying it is a semantic model class called ScreenAction that does the actual work of building all the constituent WPF objects. That class looks like this:

```
public interface IScreenAction
{
    bool IsPermanent { get; set; }
    InputBinding Binding { get; set; }
    string Name { get; set; }
    Icon Icon { get; set; }
    ICommand Command { get; }
    bool ShortcutOnly { get; set; }
    void BuildButton(ICommandBar bar);
}
```

This is an important detail. I can build and test the ScreenAction object independently of the fluent interface, and now the fluent interface merely has to configure ScreenAction objects. The actual DSL is implemented on a class called ScreenObjectRegistry that tracks the list of active ScreenAction objects (see **Figure 1**).

The registration of a new screen action begins with the call to the Action(name) method above and returns a new instance of the BindingExpression class that acts as an Expression Builder to configure the new ScreenAction object, partially shown in **Figure 2**.

One of the important factors in many fluent interfaces is trying to guide the user of the API into doing things in a certain order. In the case in **Figure 2**, I use interfaces on BindingExpression strictly to control the user choices in IntelliSense, even though I am always returning the same BindingExpression object throughout. Think about this. Users of this fluent interface should only specify the action name and the keyboard shortcut keys once. After that, the user shouldn't have to see those methods in IntelliSense. The DSL expression starts with the call to ScreenObjectRegistry.Action(name), which captures the descriptive name of the shortcut that will appear in menus and returns a new BindingExpression object as this interface:

```
public interface IActionExpression
{
    IBindingExpression Bind(Key key);
    IBindingExpression Bind(ModifierKeys modifiers, Key key);
}
```

By casting BindingExpression to IActionExpression, the only choice the user has is to specify the key combinations for the shortcut, which will return the same BindingExpression object, but casted to the IBindingExpression interface that only allows users to specify a single action:

```
// The last step that captures the actual
// "action" of the ScreenAction
public interface IBindingExpression
{
    ScreenAction ToDialog<T>();
    ScreenAction ToScreen<T>() where T : IScreen;
    ScreenAction PublishEvent<T>() where T : new();
    ScreenAction To(Action action);
    ScreenAction To(ICommand command);
}
```

Object Initializers

Now that we've introduced method chaining as the mainstay of internal DSL development in C#, let's start looking at the alternative patterns that can often lead to simpler mechanics for the DSL developer. The first alternative is simply to use the object initializer functionality introduced in the Microsoft .NET Framework 3.5.

I can still remember my very first foray into fluent interfaces. I worked on a system that acted as a message broker between law firms submitting legal invoices electronically and their customers. One of the common use cases for us was to send messages to the customers on behalf of the law firms. To send the messages we invoked an interface like this:

```
public interface IMessageSender
{
    void SendMessage(string text, string sender, string receiver);
}
```

That's a very simple API; just pass in three string arguments and it's good to go. The problem in usage is which argument goes where. Yes, tools such as ReSharper can show you which parameter you're specifying at any one time, but how about scanning the calls to SendMessage when you're just reading code? Look at the usage of the following code sample and you'll understand exactly what I mean about errors from transposing the order of the string arguments:

```
// Snippet from a class that uses IMessageSender
public void SendMessage(IMessageSender sender)
{
    // Is this right?
    sender.SendMessage("the message body", "PARTNER001", "PARTNER002");

    // or this?
    sender.SendMessage("PARTNER001", "the message body", "PARTNER002");

    // or this?
    sender.SendMessage("PARTNER001", "PARTNER002", "the message body");
}
```

At the time, I solved the API usability issue by moving to a fluent interface approach that more clearly indicated which argument was which:

```
public void SendMessageFluently(FluentMessageSender sender)
{
    sender
        .SendText("the message body")
        .From("PARTNER001").To("PARTNER002");
}
```

I genuinely believed this made for a more usable, less error-prone API, but let's look at what the underlying implementation of the expression builders might look like in **Figure 3**.

That's a lot more code to create the API than was originally required. Fortunately, now we have another alternative with object initializers (or with named parameters in .NET Framework 4 or VB.NET). Let's make another version of the message sender that takes in a single object as its parameter:

```
public class SendMessageRequest
{
    public string Text { get; set; }
    public string Sender { get; set; }
    public string Receiver { get; set; }
}

public class ParameterObjectMessageSender
{
    public void Send(SendMessageRequest request)
    {
        // send the message
    }
}
```

Figure 2 **BindingExpression Class Acting as Expression Builder**

```
public class BindingExpression : IBindingExpression, IActionExpression
{
    private readonly ScreenObjectRegistry _registry;
    private readonly ScreenAction _screenAction = new ScreenAction();
    private KeyGesture _gesture;

    public BindingExpression(string name, ScreenObjectRegistry registry)
    {
        _screenAction.Name = name;
        _registry = registry;
    }

    public IBindingExpression Bind(Key key)
    {
        _gesture = new KeyGesture(key);
        return this;
    }

    public IBindingExpression Bind(ModifierKeys modifiers, Key key)
    {
        _gesture = new KeyGesture(key, modifiers);
        return this;
    }

    // registers an ICommand that will launch the dialog T
    public ScreenAction ToDialog<T>()
    {
        return buildAction(() => _registry.CommandForDialog<T>());
    }

    // registers an ICommand that would open the screen T in the
    // main tab area of the UI
    public ScreenAction ToScreen<T>() where T : IScreen
    {
        return buildAction(() => _registry.CommandForScreen<T>());
    }

    public ScreenAction To(ICommand command)
    {
        return buildAction(() => command);
    }

    // Merely configures the underlying ScreenAction
    private ScreenAction buildAction(Func<ICommand> value)
    {
        ICommand command = value();
        _screenAction.Binding = new KeyBinding(command, _gesture);

        _registry.register(_screenAction);

        return _screenAction;
    }

    public BindingExpression Icon(Icon icon)
    {
        _screenAction.Icon = icon;
        return this;
    }
}
```

Now, the API usage with an object initializer is:

```
public void SendMessageAsParameter(ParameterObjectMessageSender sender)
{
    sender.Send(new SendMessageRequest()
    {
        Text = "the message body",
        Receiver = "PARTNER001",
        Sender = "PARTNER002"
    });
}
```

Arguably, this third incarnation of the API reduces errors in usage with much simpler mechanics than the fluent interface version.

The point here is that fluent interfaces are not the only pattern for creating more readable APIs in the .NET Framework. This approach is much more common in JavaScript, where you can

Figure 3 **Implementation of an Expression Builder**

```
public class FluentMessageSender
{
    private readonly IMessageSender _messageSender;

    public FluentMessageSender(IMessageSender sender)
    {
        _messageSender = sender;
    }

    public SendExpression SendText(string text)
    {
        return new SendExpression(text, _messageSender);
    }

    public class SendExpression : ToExpression
    {
        private readonly string _text;
        private readonly IMessageSender _messageSender;
        private string _sender;

        public SendExpression(string text, IMessageSender messageSender)
        {
            _text = text;
            _messageSender = messageSender;
        }

        public ToExpression From(string sender)
        {
            _sender = sender;
            return this;
        }

        void ToExpression.To(string receiver)
        {
            _messageSender.SendMessage(_text, _sender, receiver);
        }
    }

    public interface ToExpression
    {
        void To(string receiver);
    }
}
```

use JavaScript Object Notation (JSON) to completely specify objects in one line of code, and in Ruby, where it is idiomatic to use name/value hashes as arguments to methods.

Nested Closure

I think that many people assume that fluent interfaces and method chaining are the only possibilities for building DSLs inside C#. I used to believe that too, but I've since found other techniques and patterns that are frequently much easier to implement than method chaining. An increasingly popular pattern is the nested closure pattern:

Express statement sub-elements of a function call by putting them into a closure in an argument.

More and more .NET Web development projects are being done with the Model-View-Controller pattern. One of the side effects of this shift is much more need to generate snippets of HTML in code for input elements. Straight-up string manipulation to generate the HTML can get ugly fast. You end up repeating a lot of calls to "sanitize" the HTML to avoid injection attacks, and in many cases we may want to allow multiple classes or methods to have some say in the final HTML representation. I want to express HTML creation by just saying "I want a div tag with this text and this class." To ease this HTML generation, we model HTML with an "HtmlTag" object that looks something like this in usage

```
var tag = new HtmlTag("div").Text("my text").AddClass("collapsible");
Debug.WriteLine(tag.ToString());
```

which generates the following HTML

```
<div class="collapsible">my text</div>
```

The core of this HTML generation model is the HtmlTag object that has methods to programmatically build up an HTML element structure like this:

```
public interface IHtmlTag
{
    HtmlTag Attr(string key, object value);
    HtmlTag Add(string tag);
    HtmlTag AddStyle(string style);
    HtmlTag Text(string text);
    HtmlTag SetStyle(string className);
    HtmlTag Add(string tag, Action<HtmlTag> action);
}
```

This model also allows us to add nested HTML tags like this:

```
[Test]
public void render_multiple_levels_of_nesting()
{
    var tag = new HtmlTag("table");
    tag.Add("tbody/tr/td").Text("some text");

    tag.ToCompacted().ShouldEqual(
       "<table><tbody><tr><td>some text</td></tr></tbody></table>"
    );
}
```

In real usage, I frequently find myself wanting to add a fully configured child tag in one step. As I mentioned, I have an open source project called StoryTeller that my team is using to express acceptance tests. Part of the functionality of StoryTeller is to run all of the acceptance tests in our continuous integration build and generate a report of the test results. The test result summary is expressed as a simple table with three columns. The summary table HTML looks like this:

```
<table>
    <thead>
        <tr>
            <th>Test</th>
            <th>Lifecycle</th>
            <th>Result</th>
        </tr>
    </thead>
    <tbody>
        <!-- rows for each individual test -->
    </tbody>
</table>
```

IronRuby is exceptional for creating internal DSLs because of its flexible and relatively clutter-free syntax.

Using the HtmlTag model I described above, I generate the header structure of the results table with this code:

```
// _table is an HtmlTag object

// The Add() method accepts a nested closure argument
_table.Add("thead/tr", x =>
{
    x.Add("th").Text("Test");
    x.Add("th").Text("Lifecycle");
    x.Add("th").Text("Result");
});
```

In the call to _table.Add I pass in a lambda function that completely specifies how to generate the first header row. Using the nested closure pattern allows me to pass in the specification without first having to create another variable for the "tr" tag. You might not like this syntax at first glance, but it makes the code terser. Internally, the Add method that uses the nested closure is simply this:

```
public HtmlTag Add(string tag, Action<HtmlTag> action)
{
    // Creates and adds the new HtmlTag with
    // the supplied tagName
    var element = Add(tag);

    // Uses the nested closure passed into this
    // method to configure the new child HtmlTag
    action(element);

    // returns that child
    return element;
}
```

For another example, the main StructureMap Container class is initialized by passing in a nested closure that represents all of the desired configuration for the container like this:

```
IContainer container = new Container(r =>
{
    r.For<Processor>().Use<Processor>()
        .WithCtorArg("name").EqualTo("Jeremy")
        .TheArrayOf<IHandler>().Contains(x =>
        {
            x.OfConcreteType<Handler1>();
            x.OfConcreteType<Handler2>();
            x.OfConcreteType<Handler3>();
        });
});
```

The signature and body of this constructor function is:

```
public Container(Action<ConfigurationExpression> action)
{
    var expression = new ConfigurationExpression();
    action(expression);

    // As explained later in the article,
    // PluginGraph is part of the Semantic Model
    // of StructureMap
    PluginGraph graph = expression.BuildGraph();

    // Take the PluginGraph object graph and
    // dynamically emit classes to build the
    // configured objects
    construct(graph);
}
```

I used the nested closure pattern in this case for a couple of reasons. The first is that the StructureMap container works by taking the complete configuration in one step, then using Reflection.Emit to dynamically generate "builder" objects before the container can be used. Taking the configuration in through a nested closure allows me to capture the entire configuration at one time and quietly do the emitting right before the container is made available for use. The other reason is to segregate the methods for registering types with the container at configuration time away from the methods that you would use at runtime to retrieve services (this is an example of the Interface Segregation Principle, the "I" in S.O.L.I.D.).

I have included the nested closure pattern in this article because it's becoming quite prevalent in.NET Framework open source projects such as Rhino Mocks, Fluent NHibernate and many IoC tools. Also, I have frequently found the nested closure pattern to be significantly easier to implement than using only method chaining. The downside is that many developers are still uncomfortable with lambda expressions. Furthermore, this technique is barely usable in VB.NET because VB.NET doesn't support multiline lambda expressions.

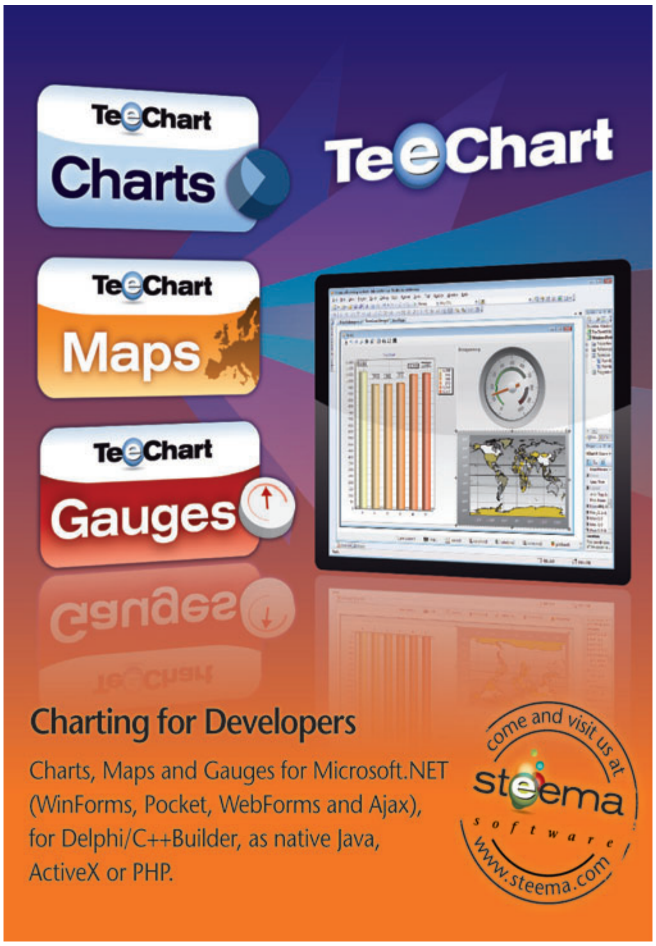
IronRuby and Boo

All of my samples in this article are written in C# for mainstream appeal, but if you're interested in doing DSL development you may want to look at using other CLR languages. In particular, IronRuby is exceptional for creating internal DSLs because of its flexible and relatively clutter-free syntax (optional parentheses, no semicolons and very terse). Stepping farther afield, the Boo language is also popular for DSL development in the CLR.

The design pattern names and definitions are taken from the online draft of Martin Fowler's forthcoming book on Domain Specific Languages at martinfowler.com/dslwip/index.html. ■

JEREMY MILLER, *a Microsoft MVP for C#, is also the author of the open source StructureMap (structuremap.sourceforge.net) tool for Dependency Injection with .NET and the forthcoming StoryTeller (storyteller.tigris.org) tool for supercharged FIT testing in .NET. Visit his blog, The Shade Tree Developer, at codebetter.com/blogs/jeremy.miller, part of the CodeBetter site.*

THANKS to the following technical expert for reviewing this article: *Glenn Block*


TeeChart Charts
TeeChart Maps
TeeChart Gauges
TeeChart
Charting for Developers
Charts, Maps and Gauges for Microsoft.NET (WinForms, Pocket, WebForms and Ajax), for Delphi/C++Builder, as native Java, ActiveX or PHP.
come and visit us at
steema software
www.steema.com

WINDOWS 7 IS HERE... VISUAL STUDIO 2010 IS COMING
PREPARE FOR THE WORLD OF
TOMORROW
WITH STUDIO ENTERPRISE 2009 V3

100'S OF CONTROLS, 7 PLATFORMS, FEATURING

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

Grids • Charts • Reports • Schedules • Menus • Toolbars • Ribbon • Data Input • Editors • PDF

DEVELOPER APPROVED FOR 20+ YEARS!

General Checking $1108.50

Location	Deposits	Withdrawls
NEW YORK CITY, NY	3307	
DUBAI, INDIA		1593
PITTSBURGH, PA		3977
BEIJING, CHINA		4745
DENVER, CO		3294
CHICAGO, IL	3587	
PARIS, FRANCE		4222

Savings	$11108.50
	$3108.30
Funds	$9104.10
cy Funds	$5108.80
	$4878.70

DOWNLOAD YOUR FREE TRIAL AT

WWW.COMPONENTONE.COM/PREPARE

ComponentOne   **ComponentOne Sales:** 1.800.858.2739 or 1.412.681.4343

Designing Services for Windows Azure

Thomas Erl, Arman Kurtagic and Herbjörn Wilhelmsen

Windows Azure is a new cloud computing platform under development by Microsoft (microsoft.com/windowsazure). Cloud computing allows developers to host applications in an Internet-accessible virtual environment. The environment transparently provides the hardware, software, network and storage needed by the application.

As with other cloud environments, Windows Azure provides a hosted environment for applications. The added benefit of Windows Azure is that .NET Framework applications can be deployed with minimal changes from their desktop siblings.

Applying service-oriented architecture (SOA) patterns and utilizing the experiences collected when implementing service-oriented solutions will be key to success when moving your services and applications into the new arena of cloud computing. To better understand how SOA patterns can be applied to Windows Azure deployments, let's take a look at a scenario in which a fictional bank moves its services to the cloud.

Disclaimer: This article is based on a pre-release version of Windows Azure. All information is subject to change.

This article discusses:

- Deployment issues
- Designing for performance and flexibility
- Using messages and queues
- Idempotent capability

Technologies discussed:

Windows Azure

Code download available at:

code.msdn.microsoft.com/mag201001Azure

Cloud Banking

Woodgrove Bank is a small financial institution that has decided to focus on a new online banking initiative branded Woodgrove Bank Online. One of Woodgrove Bank's most important clients, Fourth Coffee, volunteered to try out the new solution for processing card transactions. A subset of the services planned for the solution is already live, and the availability of these services has generated more interest from other customers. However, as more of the solution's rollout is planned, challenges emerge.

The first issue pertains to scalability and reliability. Woodgrove Bank never wanted to take responsibility for hosting its IT solutions. Instead, it established a provisioning agreement with a local ISP called the Sesame Hosting Company. To date, Sesame Hosting has fulfilled the Web hosting needs of Woodgrove Bank, but the new card-processing solution has introduced scalability requirements that Sesame Hosting is not prepared to handle.

The Woodgrove Bank technology architecture team suggests redundantly deploying the Woodgrove Bank Online services, as per the Redundant Implementation pattern (descriptions of the patterns discussed here can be found at soapatterns.org). In essence,

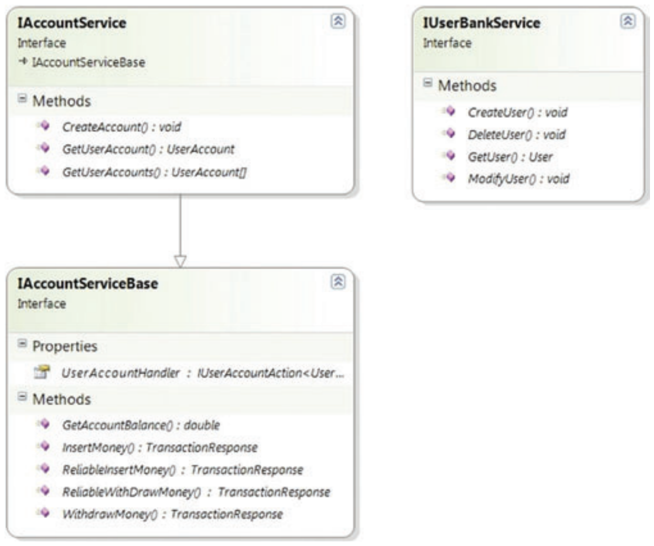

IAccountService
Interface
→ IAccountServiceBase
Methods
CreateAccount() : void
GetUserAccount() : UserAccount
GetUserAccounts() : UserAccount[]
IUserBankService
Interface
Methods
CreateUser() : void
DeleteUser() : void
GetUser() : User
ModifyUser() : void
IAccountServiceBase
Interface
Properties
UserAccountHandler : IUserAccountAction<User...
Methods
GetAccountBalance() : double
InsertMoney() : TransactionResponse
ReliableInsertMoney() : TransactionResponse
ReliableWithDrawMoney() : TransactionResponse
WithdrawMoney() : TransactionResponse

Figure 1 **The Initial Service Contracts**

the pattern suggests an approach whereby services are intentionally deployed redundantly for increased scalability and failover. The Sesame Hosting company investigates this option, but cannot afford to expand its infrastructure in order to accommodate redundant service deployments. It simply doesn't have the resources or budget to handle the increase in hardware, operational software maintenance and networking appliances that would be required.

The time frame is also a problem. Even if Sesame Hosting could make the necessary infrastructure available, it could not do so in time for Woodgrove Bank to meet its planned rollout schedule. The need to hire and train personnel alone would prolong the infrastructure expansion far beyond Woodgrove Bank's timetable.

After realizing that Sesame Hosting wouldn't be able to meet its needs, the Woodgrove Bank team begins to explore the option of hosting its services in a public cloud. The Windows Azure platform provides a way of virtualizing services that naturally apply the Redundant Implementation pattern. This feature of Windows Azure is called On-Demand Application Instance (discussed in the May 2009 issue at msdn.microsoft.com/magazine/dd727504). This feature, and the ability to use Microsoft datacenters without a long-term commitment, looks promising to the Woodgrove Bank team. Let's take a closer look at how Woodgrove Bank migrates its solution to Windows Azure.

Deployment Basics

The first order of business is to deploy a Web service by following a contract-first approach that adheres to the Standardized Service Contract principle. The team uses the WSCF.blue tool (msdn.microsoft.com/magazine/ee335699) to generate Windows Communication Foundation (WCF) contracts from WSDL and XSDs that were modeled for optimal interoperability. The service contracts are shown in **Figure 1**.

Because services will need to change and evolve over time, the developers also decide to let their data contracts implement the IExtensibleObject interface in support of the Forward Compatibility pattern (see **Figure 2**).

To store the necessary data, the Woodgrove Bank team wants to use SQL Azure because it already has an existing database structure that the team wants to retain. If the developers were able to use a non-relational store, they might consider Windows Azure Storage instead.

Woodgrove Bank architects proceed to create a Visual Studio Template Cloud Service and use Visual Studio to publish it. They then log onto the Windows Azure portal to create their new cloud service (see **Figure 3**).

Next, they are presented with a screen that allows them to start deploying the service. They click the Deploy button and specify an application package, configuration settings and a deployment name. After a few more clicks, their service is residing in the cloud.

Figure 4 shows an example of the service configuration.

The key to making the solution elastic with regard to Woodgrove Bank's scalability requirements is the following configuration element:

```
<Instances count="1" />
```

For example, if the developers want 10 instances, this element would be set to:

```
<Instances count="10" />
```

Figure 5 shows the screen that confirms that only one instance is up and running. Clicking the Configure button brings up a screen where they are able to edit the service configuration and change the Instances setting as required.

Performance and Flexibility

After some stress testing, the Woodgrove Bank development team found that having only one central data store in SQL Azure led to slower and slower response times when traffic increased. The developers decided to address this performance issue by using Windows Azure table storage, which is designed to improve scalability by distributing the partitions across many storage nodes. Windows Azure table storage also provides fast data access because the system monitors usage of the partitions and automatically load-balances them. However, because Windows Azure table storage isn't

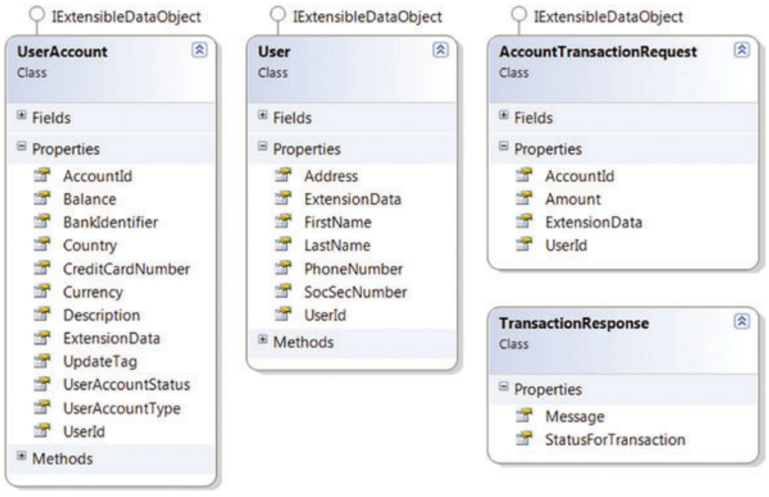


Figure 2 **The Initial Data Contracts**

a relational data store, the team had to design some new data storage structures and pick a combination of partition and row keys that would provide good response times.

They ended up with three tables as shown in **Figure 6**. UserAccountBalance will store user account balances. AccountTransactionLogg will be used for storing all transaction messages for specific accounts. The UserAccountTransaction table will be used for storing account transactions. The partition keys for the UserAccountTransaction and AccountTransactionLogg tables were created by concatenating UserId and AccountId because these are a part of all queries and can give quick response times. The partition key for the UserAccountBalance table is UserId and the row key is AccountId. Together they provide a unique identification of a user and his account.

Windows Azure table storage is designed to improve scalability by distributing the partitions across many storage nodes.

Woodgrove Bank considers the project a success thus far and wants more customers to start using the solution. Soon World Wide Importers is ready to join in—though with some new functional requirements.

The request that appears to matter most is that the service interface (or information structure) should be changed. According to World Wide Importers, the information structure that Woodgrove Bank uses is not compatible with theirs. Due to the importance of this particular customer, the Woodgrove Bank development team suggests the application of the Data Model Transformation pattern. The developers would create several new services with the interfaces that World Wide Importers requested and these services would contain logic to translate the requests between the World Wide Importers data models and the Woodgrove Bank data models.

Figure 4 **Windows Azure Service Configuration**

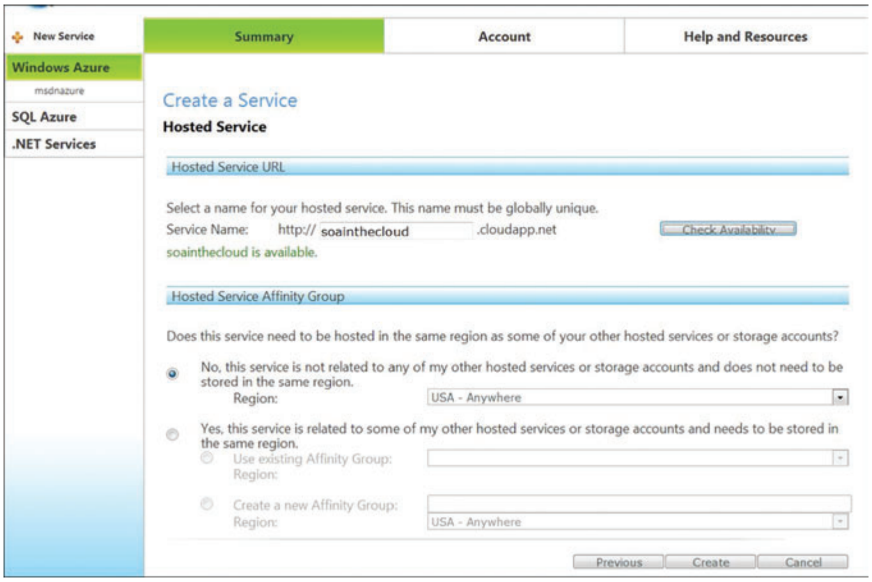
```
<Role name="BankWebRole">
  <Instances count="1" />
  <ConfigurationSettings>
    <Setting
      name="DataConnectionString"
      value="DefaultEndpointsProtocol=https;AccountName=YOURACCOUNTNAME;A
ccountKey=YOURKEY" />
    <Setting
      name="DiagnosticsConnectionString"
      value="DefaultEndpointsProtocol=https;AccountName=YOURDIAGNOSTICSAC
COUNTNAME;AccountKey=YOURKEY" />
```

To satisfy this requirement, a new structure for the UserAccount is created. The developers are careful to ensure that there is a clear mapping between the UserAccountWwi and UserAccount classes, as shown in **Figure 7**.

Service contracts need to accept a specific data contract (UserAccountWwi) that transforms requests to UserAccount before passing on the call to other parts of the solution, and then transform it back in the reply. The architects at Woodgrove Bank realize that they could reuse a base service interface when implementing these new requirements. The final design is shown in **Figure 8**.

The developers choose to implement the data transformations by creating a couple of extension methods for the UserAccount class, including the methods TransformToUserAccountWwi and TransformToUserAccount.

The new service accepts the UserAccountWwi data contract. Prior to sending requests on to other layers, the data is transformed to UserAccount by calling the extension method TransformToUserAccount. Before sending a response back to the consumer, the UserAccount contract is transformed back to UserAccountWwi by calling the TransformToUserAccountWwi. For details about these elements see the source code for UserAccountServiceAdvanced in the code download for this article.



Create a Service
Hosted Service

Hosted Service URL

Select a name for your hosted service. This name must be globally unique.
Service Name: http:// soainthecloud .cloudapp.net
soainthecloud is available.

Hosted Service Affinity Group

Does this service need to be hosted in the same region as some of your other hosted services or storage accounts?

[✓ No, this service is not related to any of my other hosted services or storage accounts and does not need to be stored in the same region.]
Region: USA - Anywhere

Yes, this service is related to some of my other hosted services or storage accounts and needs to be stored in the same region.
Use existing Affinity Group:
Region:
Create a new Affinity Group:
Region: USA - Anywhere

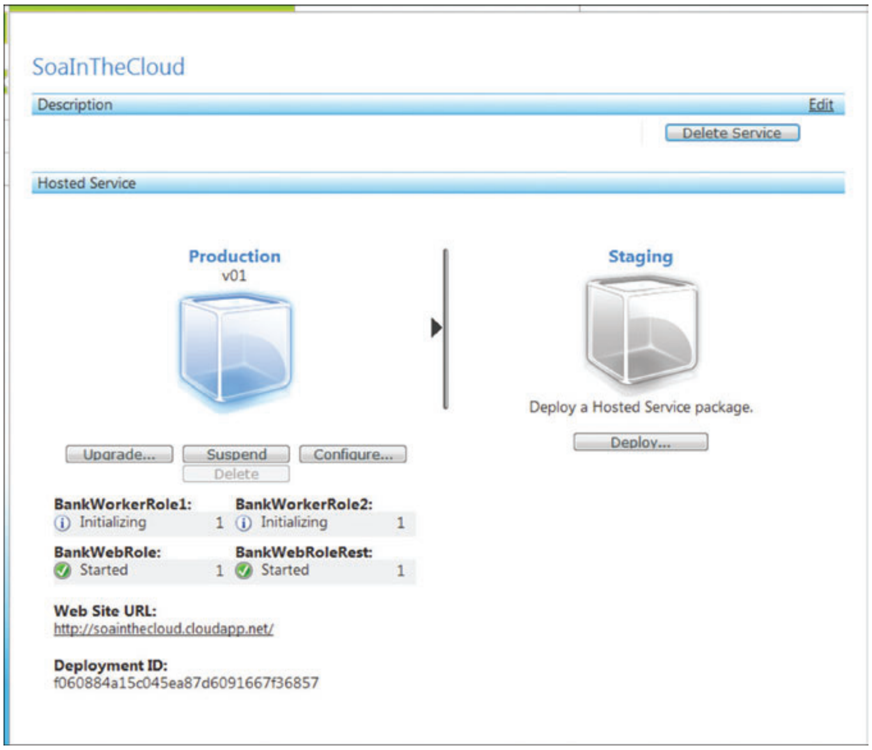
Figure 3 **Creating a Service in the Windows Azure Portal**

Messaging and Queuing

Although Woodgrove Bank is now up and running and able to facilitate a great number of incoming requests, analysts have noted significant peaks in service usage. Some of these peaks come regularly (specifically, on Monday mornings and Thursday afternoons). However, some fluctuations are unpredictable.

Putting more resources online via Windows Azure configuration would be one easy solution, but now that some large clients such as World Wide Importers are interested in the new services, concurrent usage fluctuations are expected to increase.

The developers at Woodgrove Bank took a closer look at Windows Azure offerings and



BankWorkerRole1: Initializing 1
BankWorkerRole2: Initializing 1
BankWebRole: Started 1
BankWebRoleRest: Started 1

Web Site URL:
http://soainthecloud.cloudapp.net/

Deployment ID:
f060884a15c045ea87d6091667f36857

Figure 5 **Instances Running in Windows Azure**

discovered features that allow for the application of the Reliable Messaging and Asynchronous Queuing patterns. They concluded that Reliable Messaging was not the most suitable choice as it would restrict their customer's technical choices. Asynchronous Queuing requires no special technology from the customers so they would focus on that. Inside the Windows Azure cloud, however, Reliable Messaging made perfect sense since the technology used there was all provided by Microsoft.

The objective is that no message should be lost even if services are offline due to error conditions or planned maintenance. The Asynchronous Queuing pattern allows this, though some offerings are not suitable for this pattern. For example, prompt answers with confirmation or denial of money transfers are necessary when dealing with online card transactions. But in other situations the pattern would do fine.

Communication between the Web and Worker roles (see msdn.microsoft.com/magazine/dd727504 for an explanation of these roles) is done with Windows Azure Queues (as of the November CTP version it is possible to communicate directly between role instances), which are by default both asynchronous and reliable. This doesn't automatically mean that the communication between the end user and Woodgrove Bank's services is reliable. In fact, the lines of communication between the client and the services residing in the Web role are clearly unreliable. The Woodgrove Bank team decided not to address this because implementing reliability mechanisms all the way down to the customers would in practice require customers to adhere to the same technological choices as Woodgrove Bank. This was considered unrealistic and undesirable.

Putting Queues to Work

As soon as a customer sends a message to UserAccountService, this message is placed in a Windows Azure Queue and the customer receives a confirmation message. UserAccountWorker will then be able to get the message from the queue. Should UserAccountWorker be down, the message will not be lost as it is stored securely in the queue.

If the processing inside UserAccountWorker goes wrong, the message will not be removed from the queue. To ensure this, the call to the DeleteMessage method of the queue is made only after the work has been completed. If UserAccountWorker didn't finish processing the message before the timeout elapsed (the timeout is hardcoded to 20 seconds), the message will again be made visible on the queue so that another instance of UserAccountWorker can attempt to process it.

As soon as a customer sends a message to UserAccountService, this message is placed in a queue and the customer receives a confirmation message of type TransactionResponse. From the perspective of the customer, Asynchronous Queuing is used. ReliableMessaging is used to communicate between UserAccountStorageAction and AccountStorageWorker, which reside in the Web role and Worker role, respectively. Here's how the call handler put messages into the queue:

```
public TransactionResponse ReliableInsertMoney(
  AccountTransactionRequest accountTransactionrequest) {

//last parameter (true) means that we want to serialize
//message to the queue as XML (serializeAsXml=true)
  return UserAccountHandler.ReliableInsertMoney(
    accounttransactionRequest.UserId,
    accounttransactionRequest.AccountId,
    accounttransactionRequest.Amount, true);
}
```

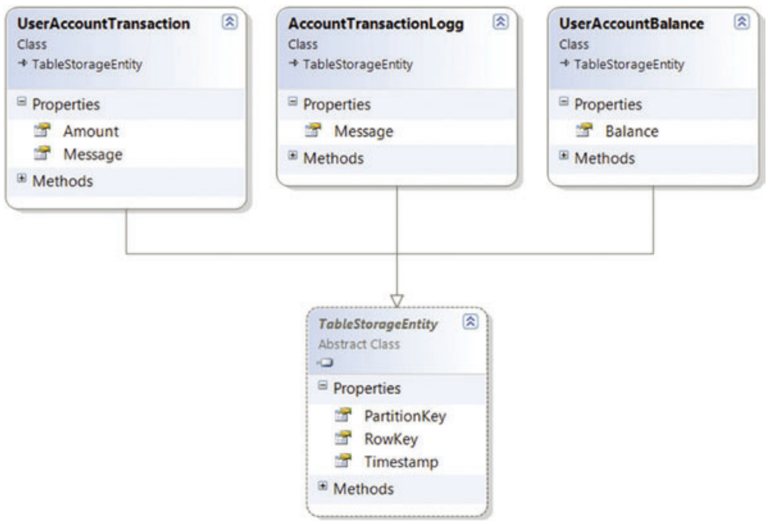


Figure 6 **Windows Azure Table Storage Models**

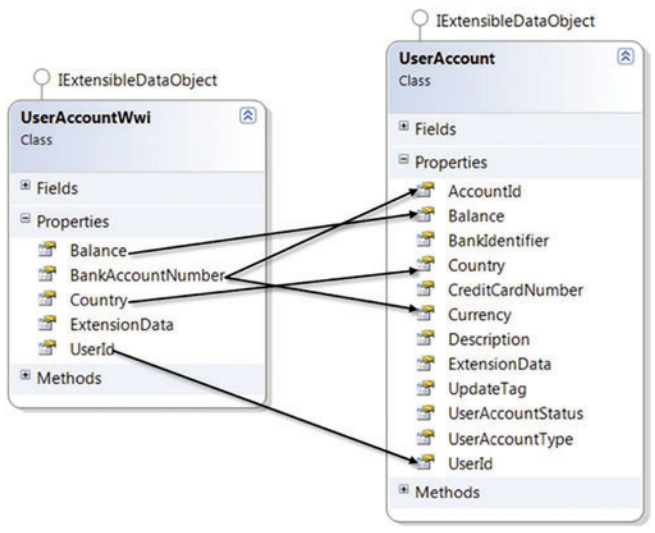


Figure 7 **UserAccount Structure for Data Model Transformation**

UserAccountHandler is a property that returns an IUserAccountAction, which is injected in the runtime. This makes it easier to separate implementation from the contract and later change the implementation:

```
public IUserAccountAction<Models.UserAccount> UserAccountHandler
  {get;set;}

public UserAccountService(
  IUserAccountAction<Models.UserAccount> action) {

  UserAccountHandler = action;
}
```

After the message is sent to one of the responsible actions, it will be put in the queue. The first method in **Figure 9** shows how data can be stored as serializable XML and the second method shows how data can be stored as a string in the queue. Note that there is a limitation in Windows Azure Queues where the maximum message size is 8KB.

The QueueManager class will initialize queues using definitions from the configuration:

```
CloudQueueClient queueClient =
  CloudStorageAccount.FromConfigurationSetting(
    "DataConnectionString").CreateCloudQueueClient();

accountTransQueue = queueClient.GetQueueReference(
  Helpers.Queues.AccountTransactionsQueue);
accountTransQueue.CreateIfNotExist();

loggQueue = queueClient.GetQueueReference(
  Helpers.Queues.AccountTransactionLoggQueue);
loggQueue.CreateIfNotExist();
```

AccountStorageWorker listens for the messages on AccountTransactionQueue and gets the messages from the queue. To be able to listen for the message, the worker must open the correct queue:

```
var storageAccount = CloudStorageAccount.FromConfigurationSetting(
  "DataConnectionString");
// initialize queue storage
CloudQueueClient queueStorage = storageAccount.CreateCloudQueueClient();
accountTransactionQueue = queueStorage.GetQueueReference(
  Helpers.Queues.AccountTransactionsQueue);
```

After the queue is opened and AccountStorageWorker reads the message, the message will be invisible in the queue for 20 seconds (the visibility timeout was set to 20). During that time the worker will try to process the message.

If processing of the message succeeds, the message will be deleted from the queue. If processing fails, the message will be put back in the queue.

Processing Messages

The ProcessMessage method first needs to get the content of the message. This can be done in one of two ways. First, the message could be stored as a string in the queue:

```
//userid|accountid|transactionid|amount
var str = msg.AsString.Split('|');...
```

Second, the message could be serialized XML:

```
using (MemoryStream m =
  new MemoryStream(msg.AsBytes)) {

  if (m != null) {
    XmlSerializer xs = new XmlSerializer(
      typeof(Core.TableStorage.UserAccountTransaction));
    var t = xs.Deserialize(m) as
      Core.TableStorage.UserAccountTransaction;

    if (t != null) { ....... }
  }
}
```

Should the AccountStorageWorker for some reason be down or unable to process the message, no message will be lost as it is saved in the queue. If processing inside the AccountStorageWorker should fail, the message will not be removed from the queue and it will become visible in the queue after 20 seconds.

To ensure this behavior, the call to the DeleteMessage method of the queue is made only after the work has been completed. If AccountStorageWorker didn't finish processing the message before the timeout elapsed, the message will yet again be made visible on the queue so that another instance of AccountStorageWorker can attempt processing it. **Figure 10** works on a message that was stored as a string.

Idempotent Capability

What if one of Woodgrove Bank's customers sends a request to transfer money from one account to another and the message gets lost? If the customer resends the message, it is possible that two or more of the requests reach the services and gets treated separately.

One of the Woodgrove Bank team members immediately identified this scenario as one that requires the Idempotent Capability pattern. This pattern demands that capabilities or operations are implemented in such a way that they are safe to repeat. In short, the solution

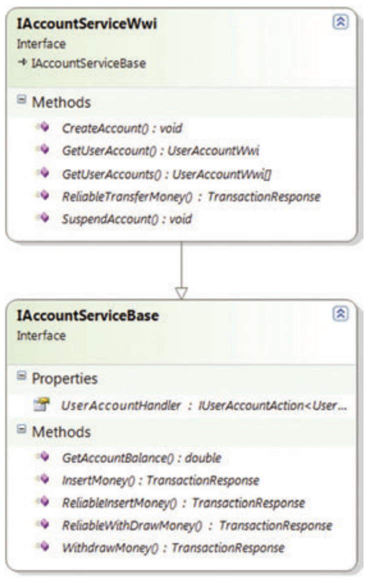

IAccountServiceWwi
Interface
IAccountServiceBase
Methods
CreateAccount() : void
GetUserAccount() : UserAccountWwi
GetUserAccounts() : UserAccountWwi[]
ReliableTransferMoney() : TransactionResponse
SuspendAccount() : void
IAccountServiceBase
Interface
Properties
UserAccountHandler : IUserAccountAction<User...
Methods
GetAccountBalance() : double
InsertMoney() : TransactionResponse
ReliableInsertMoney() : TransactionResponse
ReliableWithDrawMoney() : TransactionResponse
WithdrawMoney() : TransactionResponse

Figure 8 **Service Contracts for World Wide Importers**

Figure 9 **Storing Data**

```
public TransactionResponse ReliableHandleMoneyInQueueAsXml(
  UserAccountTransaction accountTransaction){

  using (MemoryStream m = new MemoryStream()){
    XmlSerializer xs =
      new XmlSerializer(typeof(UserAccountTransaction));
    xs.Serialize(m, accountTransaction);

    try
    {
      QueueManager.AccountTransactionsQueue.AddMessage(
        new CloudQueueMessage(m.ToArray()));
      response.StatusForTransaction = TransactionStatus.Succeded;
    }
    catch(StorageClientException)
    {
      response.StatusForTransaction = TransactionStatus.Failed;
      response.Message =
        String.Format("Unable to insert message in the account
transaction queue userId|AccountId={0}, messageId={1}",
        accountTransaction.PartitionKey, accountTransaction.RowKey);
    }
  }
  return response;
}

public TransactionResponse ReliableHandleMoneyInQueue(
  UserAccountTransaction accountTransaction){

  TransactionResponse response = this.CheckIfTransactionExists(
    accountTransaction.PartitionKey, accountTransaction.RowKey);

  if (response.StatusForTransaction == TransactionStatus.Proceed)
  {
    //userid|accountid is partkey
    //userid|accountid|transactionid|amount
    string msg = string.Format("{0}|{1}|{2}",
      accountTransaction.PartitionKey,
      accountTransaction.RowKey,
      accountTransaction.Amount);

    try
    {
      QueueManager.AccountTransactionsQueue.AddMessage(
        new CloudQueueMessage(msg));
      response.StatusForTransaction = TransactionStatus.Succeded;
    }
    catch(StorageClientException)
    {
      response.StatusForTransaction = TransactionStatus.Failed;
      response.Message =
        String.Format("Unable to insert message in the account
transaction queue userId|AccountId={0}, messageId={1}",
        accountTransaction.PartitionKey, accountTransaction.RowKey);
    }
  }
  return response;
}
```

that Woodgrove Bank wants to implement requires well-behaved clients that attach a unique ID to each request and promise that they will resend the exact same message including the same unique ID in case of a retry. To be able to handle this, the unique ID is saved in the Windows Azure table storage. Before processing any requests, it is necessary to check if a message with that ID was already processed. If it has been processed, a correct reply will be created, but the processing associated with the new request will not take place.

The objective is that no message should be lost even if services are offline due to error conditions or planned maintenance.

Although this means bothering the central data store with extra queries, it was deemed necessary. It will result in some deterioration of performance since some queries are made to the central data store before any other processing can take place. However, allowing this to consume extra time and other resources is a reasonable choice in order to meet Woodgrove Bank's requirements.

The Woodgrove Bank team updated the methods ReliableInsertMoney and ReliableWithDrawMoney in the IUserAccountAction and their implementations by adding a transaction ID:

```
TransactionResponse ReliableInsertMoney(
  Guid userId, Guid accountId, Guid transactionId,
  double amount, bool serializeToQueue);

TransactionResponse ReliableWithDrawMoney(
  Guid userId, Guid accountId, Guid transactionId,
  double amount, bool serializeToQueue);
```

The UserAccountTransaction table (Windows Azure Storage) was updated by adding TransactionId as RowKey, so that each insert into the table would have a unique transaction ID.

The responsibility for sending a unique message ID for each unique transaction is set to the client:

```
WcfClient.Using(new AccountServiceClient(), client =>{
  using (new OperationContextScope(client.InnerChannel))
  {
    OperationContext.Current.OutgoingMessageHeaders.MessageId =
      messageId;
    client.ReliableInsertMoney(new AccountTransactionRequest {
      UserId = userId, AccountId = accountId, Amount = 1000 });
  }
});
```

The helper class used here can be found at soamag.com/I32/0909-4.asp.

The IUserAccountService definition was left unchanged. The only change that is necessary to implement this functionality is to read the MessageId from the incoming message headers, which was sent by the client, and use it in the processing behind the scenes (see **Figure 11**).

Figure 10 **Handling Queued Messages**

```
if (str.Length == 4){
  //userid|accountid|transactionid|amount
  UserAccountSqlAzureAction ds = new UserAccountSqlAzureAction(
    new Core.DataAccess.UserAccountDB("ConnStr"));
  try
  {
    Trace.WriteLine(String.Format("About to insert data to DB:{0}", str),
      "Information");
    ds.UpdateUserAccountBalance(new Guid(str[0]), new Guid(str[1]),
      double.Parse(str[3]));
    Trace.WriteLine(msg.AsString, "Information");
    accountTransactionLoggQueue.DeleteMessage(msg);
    Trace.WriteLine(String.Format("Deleted:{0}", str), "Information");
  }
  catch (Exception ex)
  {
    Trace.WriteLine(String.Format(
      "fail to insert:{0}", str, ex.Message), "Error");
  }
}
```

Figure 11 **Capturing Message IDs**

```
public TransactionResponse ReliableInsertMoney(
  AccountTransactionRequest accountTransactionrequest) {
  var messageId =
    OperationContext.Current.IncomingMessageHeaders.MessageId;
  Guid messageGuid = Guid.Empty;
  if (messageId.TryGetGuid(out messageGuid))
    //last parameter (true) means that we want to serialize
    //message to the queue as XML (serializeAsXml=true)
    return UserAccountHandler.ReliableInsertMoney(
      accounttransactionRequest.UserId,
      accounttransactionRequest.AccountId, messageId,
      accounttransactionRequest.Amount, true);
  else
    return new TransactionResponse { StatusForTransaction =
      Core.Types.TransactionStatus.Failed,
      Message = "MessageId invalid" };
}
```

The updated UserAccountAction will now get a transaction ID for each idempotent operation. When the service tries to complete one idempotent operation, it will check to see if the transaction exists in the table storage. If the transaction exists, the service returns the message of the transaction that was stored in the AccountTransactionLogg table. The transaction ID will be saved as RowKey in storage table UserAccountTransaction. To find the correct user and account, the service sends the partition key (userid|accountid). If the transaction ID is not found, the message will be put in the AccountTransactionsQueue for further processing:

```
public TransactionResponse ReliableHandleMoneyInQueueAsXml(
  UserAccountTransaction accountTransaction) {
  TransactionResponse response = this.CheckIfTransactionExists(
    accountTransaction.PartitionKey, accountTransaction.RowKey);
  if(response.StatusForTransaction == TransactionStatus.Proceed) {
    ...
  }
  return response;
}
```

The CheckIfTransactionExists method (see **Figure 12**) is used to ensure that the transaction has not been processed. It will try to find the transaction ID for a specific user account. If the transaction ID is found, the client will get a response message with the details of the already completed transaction.

An interesting property of CheckIfTransactionExists is that if the data you want to find is not found, Windows Azure Storage returns a 404 HTTP status code (because it uses a REST interface). Furthermore, if the data is not found, an exception will be thrown by ADO.NET client services (System.Data.Services.Client).

The responsibility for sending a unique message ID for each unique transaction is set to the client.

More Info

For more information about the implementation of this proof-of-concept solution, please check the provided source code available online. SOA pattern descriptions are published at soapatterns.org. For questions, contact herbjorn@wilhelmsen.se. ■

ARMAN KURTAGIĆ *is a consultant focusing on new Microsoft technologies, working at Omegapoint, which provides business-driven, secure IT solutions. He has worked in various roles including, developer, architect, mentor and entrepreneur, and has experience in industries such as finance, gaming and media.*

HERBJÖRN WILHELMSEN *is a consultant working for Forefront Consulting Group and is based in Stockholm. His main focus areas are service-oriented architecture and business architecture. Wilhelmsen is the chair of the SOA Patterns Review Committee and also currently leads the Business 2 IT group within the Swedish chapter of IASA. He is co-authoring the book "SOA with .NET and Azure" as part of the "Prentice Hall Service-Oriented Computing Series from Thomas Erl."*

THOMAS ERL *is the world's top-selling SOA author, series editor of the "Prentice Hall Service-Oriented Computing Series from Thomas Erl," and editor of SOA Magazine. Erl is the founder of SOA Systems Inc. and the SOASchool.com SOA Certified Professional program. Erl is the founder of the SOA Manifesto working group and is a speaker and instructor for private and public events. For more information, visit thomaserl.com.*

THANKS to the following technical expert for reviewing this article: *Steve Marx*

Figure 12 **Checking Transaction Status and ID**

```
private TransactionResponse CheckIfTransactionExists(
  string userIdAccountId, string transId) {

  TransactionResponse transactionResponse =
    new Core.Models.TransactionResponse();

  var transaction = this.TransactionExists(userIdAccountId, transId);
  if (transaction != null) {
    transactionResponse.Message =
      String.Format("messageId:{0}, Message={1}, ",
      transaction.RowKey, transaction.Message);
    transactionResponse.StatusForTransaction =
      TransactionStatus.Completed;
  }
  else
    transactionResponse.StatusForTransaction =
      TransactionStatus.Proceed;
  return transactionResponse;
}

private UserAccountTransaction TransactionExists(
    string userIdAccountId, string transId) {
    UserAccountTransaction userAccountTransaction = null;
    using (var db = new UserAccountDataContext()) {
      try {
        userAccountTransaction =
          db.UserAccountTransactionTable.Where(
          uac => uac.PartitionKey == userIdAccountId &&
          uac.RowKey == transId).FirstOrDefault();
        userAccountTransaction.Message = "Transaction Exists";
      }
      catch (DataServiceQueryException e) {
        HttpStatusCode s;
        if (TableStorageHelpers.EvaluateException(e, out s) &&
          s == HttpStatusCode.NotFound) {
          // this would mean the entity was not found
          userAccountTransaction = null;
        }
      }
    }
    return userAccountTransaction;
  }
```

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.

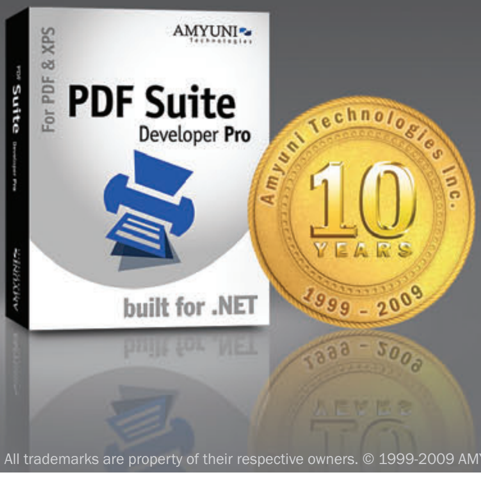
Customization

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

Fueling Your Application's Engine with Windows Azure Storage

By Kevin Hoffman and Nathan Dudek

Developers tend to cling to their physical, tangible infrastructure like a safety blanket. They know how to use it, they know how to operate it, and when something goes wrong, they know where it is. This often creates a barrier that slows down developer adoption of newer technologies, such as cloud computing.

One of the biggest questions skeptical developers ask is how they can still run background processes in the cloud—how will their *engine* continue to work. This article aims to dispel myths about lack of background processing in the cloud by showing you how you can build an application engine as well as implement asynchronous messaging and processing using Windows Azure Storage.

To prove that developers can shed the safety blanket of their physical infrastructure and put their application engines in the cloud, we're going to walk through the implementation of a small subset of an e-commerce application, Hollywood Hackers, where you can buy all of the magical technology that Hollywood uses to completely ignore the laws of physics and old-fashioned common sense.

This article discusses:

- Building an e-commerce application
- Sending asynchronous text messages ("toasts") to users
- Submitting a shopping cart to a fulfillment engine using Windows Azure technology

Technologies discussed:

Windows Azure Platform, ASP.NET MVC 2, JSON

Code Download URL:

hollywoodhackers.codeplex.com/SourceControl/ListDownloadableCommits.aspx

The two main scenarios we'll cover are:

- Sending asynchronous text messages ("toasts") to users of the application to notify them of important events such as their cart being submitted, or to send messages between employees. This scenario uses Windows Azure Queue, Windows Azure Table and a Windows Azure Worker Role.
- Submitting a shopping cart to a fulfillment engine using Windows Azure Queue and a Windows Azure Worker Role.

Figure 1 **Creating and Submitting a Message to a Windows Azure Queue**

```
string accountName;
string accountSharedKey;
string queueBaseUri;
string StorageCredentialsAccountAndKey credentials;

if (RoleEnvironment.IsAvailable)
{
// We are running in a cloud - INCLUDING LOCAL!
  accountName =
  RoleEnvironment.GetConfigurationSettingValue("AccountName");
  accountSharedKey =
  RoleEnvironment.GetConfigurationSettingValue("AccountSharedKey");
  queueBaseUri = RoleEnvironment.GetConfigurationSettingValue
 ("QueueStorageEndpoint");
}
else
{
  accountName = ConfigurationManager.AppSettings["AccountName"];
  accountSharedKey =
  ConfigurationManager.AppSettings["AccountSharedKey"];
  queueBaseUri =
  ConfigurationManager.AppSettings["QueueStorageEndpoint"];
}
credentials =
new StorageCredentialsAccountAndKey(accountName, accountSharedKey);
CloudQueueClient client =
new CloudQueueClient(queueBaseUri, credentials);
CloudQueue queue = client.GetQueueReference(queueName);
CloudQueueMessage m = new CloudQueueMessage(
  /* string or byte[] representing message to enqueue */);
Queue.AddMessage(m);
```

Intra-Application Messaging with Queue Storage

Before we get to the specific scenarios, we need to cover some basics about Windows Azure Queue. Queues in the cloud don't work quite like queues in your plain-vanilla .NET application. When you work with data in an AppDomain, you know there's only one copy of that data and it's sitting comfortably within a single managed process.

In the cloud, one piece of your data might be in California and another might be in New York, and you might have a worker role doing processing on that data in Texas and another worker role doing processing in North Dakota.

Adjusting to this kind of distributed computing and distributed data brings up issues many developers are unfamiliar with, such as coding for potential failure, building in the concept of multiple retries for data commits and, finally, the idea of idempotence.

The way Windows Azure Queues work is fairly straightforward, so long as you don't treat them like in-process regular CLR queues. First, your application will ask the queue for some number of messages (though never more than 20 at a time; keep that mind) and supply a timeout. This timeout governs how long those messages will be hidden from other queue-processing clients. When your application has successfully completed whatever processing needs to be done on the queue message, it should delete the message.

One of the biggest questions skeptical developers ask is how they can still run background processes in the cloud—how will their *engine* continue to work.

If your application throws an exception or otherwise fails to process the queue message, the message will become visible to other clients again after the timeout period. This allows additional worker roles to continue processing when one fails. Submitting a message to a queue is very straightforward: your application forms the appropriate HTTP POST message (either directly or with the aid of a client library) and submits either string or an array of bytes. Queues are designed specifically for intra-application messaging and not permanent storage, so the messages need to be kept fairly small.

As previously mentioned, you could conceivably have multiple worker roles all attempting to process the same messages. The invisibility timeout that hides messages that are currently being processed is helpful, but it isn't a guarantee. To completely avoid conflict, you should design your engine processing so that it's *idempotent*. In other words, the same queue message should be able to be processed multiple times by one or more worker roles without putting the application into an inconsistent state.

Ideally, you want the worker role to be able to detect if work has already been completed on a given message. As you write your worker roles to process queue messages, keep in mind that there is a chance your code could be attempting to process a message that already has been processed, however slim that chance may be.

Figure 2 **Storing Structured Data in the Queue**

```
namespace HollywoodHackers.Storage.Queue
{
    [Serializable]
    public class QueueMessageBase
    {
        public byte[] ToBinary()
        {
            BinaryFormatter bf = new BinaryFormatter();
            MemoryStream ms = new MemoryStream();
            ms.Position = 0;
            bf.Serialize(ms, this);
            byte[] output = ms.GetBuffer();
            ms.Close();
            return output;
        }
        public static T FromMessage<T>(CloudQueueMessage m)
        {
            byte[] buffer = m.AsBytes();
            MemoryStream ms = new MemoryStream(buffer);
            ms.Position = 0;
            BinaryFormatter bf = new BinaryFormatter();
            return (T)bf.Deserialize(ms);
        }
    }
    [Serializable]
    public class ToastQueueMessage : QueueMessageBase
    {
            public ToastQueueMessage()
            : base()
        {
        }
        public string TargetUserName { get; set; }
        public string MessageText { get; set; }
        public string Title { get; set; }
        public DateTime CreatedOn { get; set; }
   }
```

The code snippet in **Figure 1** shows how to create and submit a message to a Windows Azure Queue using the StorageClient assembly that's provided with the Windows Azure SDK. The StorageClient library is really just a wrapper around the Windows Azure Storage HTTP interface.

For other samples throughout this article, we've used some wrapper classes (available on the CodePlex site for Hollywood Hackers: hollywoodhackers.codeplex.com/SourceControl/ListDownloadableCommits.aspx) that simplify this process.

Asynchronous Messaging (Toasts)

Interactive Web sites aren't just the rage these days, they're a requirement. Users have become so accustomed to fully interactive Web sites that they think something's wrong when they encounter a static, non-interactive page. With that in mind, we want to be able to send notifications to our users as they are using the site.

To do this, we'll utilize Windows Azure Queue and Table storage mechanisms to build a message delivery framework. The client side will use jQuery combined with the jQuery Gritter plugin to display notifications in the user's browser as a toast, similar to the messages that fade in above the Windows system tray when you receive a new Outlook e-mail, instant message or tweet.

When a user needs to be sent a notification, it will be inserted into the queue. As the worker role processes each item in the queue, it will dynamically determine how to handle each one. In our case,

Figure 3 Wrapper to Interact with Queue

```
namespace HollywoodHackers.Storage.Queue
{
    public class StdQueue<T> :
    StorageBase where T : QueueMessageBase, new()
    {
        protected CloudQueue queue;
        protected CloudQueueClient client;

        public StdQueue(string queueName)
        {
            client = new CloudQueueClient
            (StorageBase.QueueBaseUri, StorageBase.Credentials);
            queue = client.GetQueueReference(queueName);
            queue.CreateIfNotExist();
        }
        public void AddMessage(T message)
        {
            CloudQueueMessage msg =
            new CloudQueueMessage(message.ToBinary());
            queue.AddMessage(msg);
        }
        public void DeleteMessage(CloudQueueMessage msg)
        {
            queue.DeleteMessage(msg);
        }
        public CloudQueueMessage GetMessage()
        {
            return queue.GetMessage(TimeSpan.FromSeconds(60));
        }
    }
    public class ToastQueue : StdQueue<ToastQueueMessage>
    {
        public ToastQueue()
            : base("toasts")
        {
        }
    }
}
```

Figure 4 **Wrapper for Table Storage**

```
namespace HollywoodHackers.Storage.Repositories
{
    public class UserTextNotificationRepository : StorageBase
    {
        public const string EntitySetName =
        "UserTextNotifications";
        CloudTableClient tableClient;
        UserTextNotificationContext notificationContext;
        public UserTextNotificationRepository()
            : base()
        {
            tableClient = new CloudTableClient
            (StorageBase.TableBaseUri, StorageBase.Credentials);
            notificationContext = new UserTextNotificationContext
            (StorageBase.TableBaseUri,StorageBase.Credentials);

            tableClient.CreateTableIfNotExist(EntitySetName);
        }
        public UserTextNotification[]
        GetNotificationsForUser(string userName)
        {
            var q = from notification in
                    notificationContext.UserNotifications
                    where notification.TargetUserName ==
                    userName select notification;
            return q.ToArray();
        }
        public void AddNotification
       (UserTextNotification notification)
        {
            notification.RowKey = Guid.NewGuid().ToString();
            notificationContext.AddObject
           (EntitySetName, notification);
            notificationContext.SaveChanges();
        }
    }
}
```

there is only one thing for the engine to do, but in a complex CRM Web site or helpdesk site, the possibilities are endless.

When the worker role comes across a user notification in the queue, it will store the notification in table storage and delete it from the queue. This allows messages to be persisted long-term and wait for the user to log in. Messages in queue storage have a short maximum lifespan and will never last more than a few days. When the user accesses the Web site, our jQuery script will asynchronously pull any messages out of the table and display them in the browser by invoking a method on a controller that returns JavaScript Object Notation (JSON) in a well-known shape.

Although the queue only handles strings or byte arrays, we can store any type of structured data in the queue by serializing it to

Queues in the cloud don't work quite like queues in your plain-vanilla .NET application.

binary and then converting it back when we need to use it. This becomes a powerful technique for passing strongly typed objects to the queue. We'll build this into the base class for our queue messages. Then our system message class can contain our data and the entire object can be submitted into the queue and utilized as needed (see **Figure 2**).

Keep in mind that in order to use the BinaryFormatter class, your Windows Azure worker role needs to be running in full-trust mode (you can enable this through your service configuration file).

Now we'll need a simple wrapper to interact with our queue. At its core, we need the ability to insert a message into the queue, get any pending messages, and clear the queue (see **Figure 3**).

We also need to set up a wrapper for our table storage so user notifications can be stored until they log in to the site. Table data is organized using a PartitionKey, which is the identifier for a *collection* of rows, and a RowKey, which uniquely identifies each individual row in a certain partition. The choice of what data you use for a PartitionKey and a RowKey could be one of the most important design decisions you make when using table storage.

These features allow load-balancing across storage nodes and provide built-in scalability options in your application. Regardless of the data-center affinity of your data, rows in table storage with the same partition key will be kept within the same physical data store. Because messages are stored for each user, the partition key will be the UserName and the RowKey will be a GUID that identifies each row (see **Figure 4**).

Now that our storage mechanisms are in place, we need a worker role that acts as our engine; processing messages in the background of our e-commerce site. To do this, we define a class that inherits from the Microsoft.ServiceHosting.ServiceRuntime.RoleEntryPoint class and associate it with the worker role in our cloud service project (see **Figure 5**).

Let's walk through the worker role code. After initializing and setting up the required queue and table storage, the code will en-

Figure 5 **Worker Role Acting as Engine**

```
public class WorkerRole : RoleEntryPoint
{
    ShoppingCartQueue cartQueue;
    ToastQueue toastQueue;
    UserTextNotificationRepository toastRepository;

    public override void Run()
    {
        // This is a sample worker implementation.
        //Replace with your logic.
        Trace.WriteLine("WorkerRole1 entry point called",
        "Information");
        toastRepository = new UserTextNotificationRepository();
        InitQueue();
        while (true)
        {
            Thread.Sleep(10000);
            Trace.WriteLine("Working", "Information");

            ProcessNewTextNotifications();
            ProcessShoppingCarts();
        }
    }
    private void InitQueue()
    {
        cartQueue = new ShoppingCartQueue();
        toastQueue = new ToastQueue();
    }
    private void ProcessNewTextNotifications()
    {
        CloudQueueMessage cqm = toastQueue.GetMessage();
        while (cqm != null)
        {
            ToastQueueMessage message =
            QueueMessageBase.FromMessage<ToastQueueMessage>(cqm);

            toastRepository.AddNotification(new
            UserTextNotification()
            {
                MessageText = message.MessageText,
                MessageDate = DateTime.Now,
                TargetUserName = message.TargetUserName,
                Title = message.Title
            });
            toastQueue.DeleteMessage(cqm);
            cqm = toastQueue.GetMessage();
        }
    }
    private void ProcessShoppingCarts()
    {
        // We will add this later in the article!
    }
    public override bool OnStart()
    {
        // Set the maximum number of concurrent connections
        ServicePointManager.DefaultConnectionLimit = 12;

        DiagnosticMonitor.Start("DiagnosticsConnectionString");
        // For information on handling configuration changes
        // see the MSDN topic at
        //http://go.microsoft.com/fwlink/?LinkId=166357.
        RoleEnvironment.Changing += RoleEnvironmentChanging;
        return base.OnStart();
    }
    private void RoleEnvironmentChanging(object sender,
RoleEnvironmentChangingEventArgs e)
    {
        // If a configuration setting is changing
        if (e.Changes.Any(change => change is
RoleEnvironmentConfigurationSettingChange))
        {
            // Set e.Cancel to true to restart this role instance
            e.Cancel = true;
        }
    }
}
```

ter a loop. Every 10 seconds, it will process messages in the queue. Each time we pass through the processing loop, we will get messages from the queue until we finally return null, indicating that we've emptied the queue.

It's worth reiterating that you can never look at more than 20 messages from the queue. Anything that does processing on a queue has a limited amount of time to do something meaningful with each queue message before the queue message is considered timed out and shows back up in the queue—making itself available for processing by other workers. Each message gets added as a user notification in table storage. An important thing to remember about worker roles is that once the entry point method finishes, that worker role is done. This is why you need to keep your logic running inside a loop.

Figure 6 **Returning Messages as JSON**

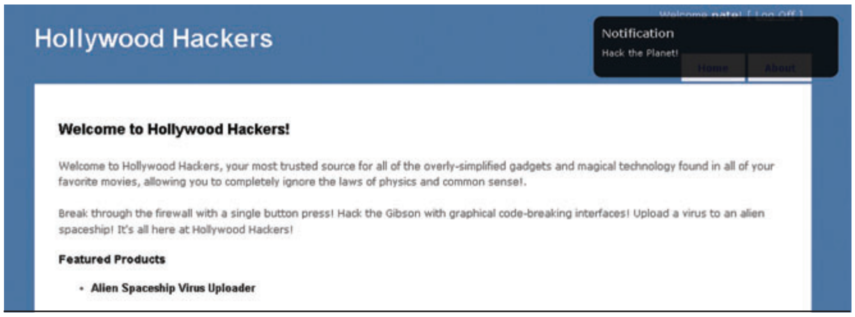
```
public JsonResult GetMessages()
{
     if (User.Identity.IsAuthenticated)
     {
UserTextNotification[] userToasts =        toastRepository.
GetNotifications(User.Identity.Name);
object[] data =
(from UserTextNotification toast in userToasts
               select new { title = toast.Title ?? "Notification",
 text = toast.MessageText }).ToArray();
            return Json(data, JsonRequestBehavior.AllowGet);
     }
     else
         return Json(null);
}
```

From the client side, we need to be able to return the messages as JSON so jQuery can asynchronously poll and display new user notifications. To do this, we'll add some code to the message controller so we can access the notifications (see **Figure 6**).

In ASP.NET MVC 2 under Visual Studio 2010 beta 2 (the environment we used to write this article), you cannot return JSON data to jQuery or any other client without the JsonRequestBehavior.AllowGet option. In ASP.NET MVC 1, this option is not necessary. Now we can write the JavaScript that will call the GetMessages method every 15 seconds and display the notifications as toast-style messages (see **Figure 7**).

Figure 7 **Notifications as Toast-Style Messages**

```
$(document).ready(function() {

    setInterval(function() {
        $.ajax({
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            url: "/SystemMessage/GetMessages",
            success: function(data) {
                for (msg in data) {
                    $.gritter.add({
                        title: data[msg].title,
                        text: data[msg].text,
                        sticky: false
                    });
                }
            }
        })
    }, 15000)
});
```


Hollywood Hackers

Welcome nate! [Log Off]

Notification

Hack the Planet!

Home About

Welcome to Hollywood Hackers!

Welcome to Hollywood Hackers, your most trusted source for all of the overly-simplified gadgets and magical technology found in all of your favorite movies, allowing you to completely ignore the laws of physics and common sense!.

Break through the firewall with a single button press! Hack the Gibson with graphical code-breaking interfaces! Upload a virus to an alien spaceship! It's all here at Hollywood Hackers!

Featured Products

- **Alien Spaceship Virus Uploader**

Figure 8 **Sample User Notification**

Submitting and Processing a Shopping Cart

For our sample application, another key scenario that we wanted to enable using queue storage was submitting shopping carts. Hollywood Hackers has a third-party fulfillment system (they can't keep all those gadgets in their little warehouse) so the engine needs to do some processing on the cart. Once the engine is finished doing its processing, it will submit a message to the user notification queue to let that user know that the shopping cart has been processed (or that something went wrong). If the user is online when the cart is processed, he will receive a pop-up toast message from the system. If he isn't online, he will receive that pop-up message the next time he logs on to the site, as shown in **Figure 8**.

What we need first are some wrapper classes to allow us to interact with the shopping cart queue. These wrappers are fairly simple and if you want to see the source code for them, you can check them out on the CodePlex site.

Unlike a standard CRUD (create, read, update, delete) repository, read operations on a queue aren't simple read operations. Remember that whenever you get a message from a queue, you've got a limited amount of time during which to process that message and either fail the operation or delete the message to indicate completed processing. This pattern doesn't translate well into the repository pattern so we've left that abstraction off the wrapper class.

Now that we have the code to interact with the shopping cart queue, we can put some code in the cart controller to submit the cart contents to the queue (see **Figure 9**).

In a real-world scenario, you would get the shopping cart from some out-of-process state like a session store, a cache or from a form post. To keep the article code simple, we're just fabricating the contents of a cart.

Finally, with the shopping cart contents sitting in the queue, we can modify our worker role so that it periodically checks the queue for pending carts. It will pull each cart out of the queue one at a time, allow itself a full minute for processing, and then submit a message to the user notification queue telling the user that the shopping cart has been processed (see **Figure 10**).

With the queue message being pulled off and put into the user notification table, the jQuery Gritter code sitting in the master page will then detect the new message on the next 15-second poll cycle and display the shopping cart toast notification to the user.

Figure 9 **Submitting Shopping Cart to Queue**

```
public ActionResult Submit()
    {
        ShoppingCartMessage cart = new ShoppingCartMessage();
        cart.UserName = User.Identity.Name;
        cart.Discounts = 12.50f;
        cart.CartID = Guid.NewGuid().ToString();
        List<ShoppingCartItem> items = new List<ShoppingCartItem>();
        items.Add(new ShoppingCartItem()
             { Quantity = 12, SKU = "10000101010",
             UnitPrice = 15.75f });
        items.Add(new ShoppingCartItem()
             { Quantity = 27, SKU = "12390123j213",
             UnitPrice = 99.92f });
        cart.CartItems = items.ToArray();
        cartQueue.AddMessage(cart);
        return View();
    }
```

Figure 10 **Checking the Queue for Pending Shopping Carts**

```
private void ProcessShoppingCarts()
{
    CloudQueueMessage cqm = cartQueue.GetMessage();

    while (cqm != null)
    {
        ShoppingCartMessage cart =
        QueueMessageBase.FromMessage<ShoppingCartMessage>(cqm);

        toastRepository.AddNotification(new UserTextNotification()
        {
            MessageText = String.Format
            ("Your shopping cart containing {0} items has been
processed.",
            cart.CartItems.Length),
            MessageDate = DateTime.Now,
            TargetUserName = cart.UserName
        });
        cartQueue.DeleteMessage(cqm);
         cqm = cartQueue.GetMessage();
    }
}
```

Summary and Next Steps

The goal of this article is to get developers to shed the safety blanket of their physical datacenters and realize that you can do more with Windows Azure than create simple "Hello World" Web sites. With the power of Windows Azure Queues and Windows Azure table storage, and using this power for asynchronous messaging between the application and its worker role(s), you truly can fuel your application's engine with Windows Azure.

To keep the article clear and easy to read, we left quite a bit of the code as is, without refactoring. As an exercise to flex your new Windows Azure muscles, try to refactor some of the code in this article to make the use of queues more generic, and even create a stand-alone assembly that contains all the code necessary to do asynchronous messaging and notifications for any ASP.NET MVC Web site.

The main thing is to roll up your sleeves, create some sites and see what you can do. The code for this article can be found on the CodePlex site for Hollywood Hackers: hollywoodhackers.codeplex.com. ■

You can find the authors blogging and ranting nonstop about new technology at exclaimcomputing.com. **KEVIN HOFFMAN** *and* **NATE DUDEK** *are the co-founders of Exclaim Computing, a company specializing in developing solutions for the cloud.*

Crypto Services and Data Security in Windows Azure

Jonathan Wiggs

Many early adopters of the Windows Azure platform still have a lot of questions about platform security and its support of cryptography. My hope here is to introduce some of the basic concepts of cryptography and related security within the Windows Azure platform. The details of this topic could fill whole books, so I am only intending to demonstrate and review some of the cryptography services and providers in Windows Azure. There are also some security implications for any transition to Windows Azure.

As with any new platform or service delivery method, you'll be faced with new challenges. You'll also be reminded that some of the classic problems still exist and even that some of the same solutions you've used in the past will still work very well. Any application engineer or designer should think about this topic as it relates to the kind of data you may be storing as well as what you need to persist. Combine this into a methodical approach and you and your customers will be well-served.

Disclaimer: This article is based on a pre-release version of Windows Azure. All information is subject to change.

This article discusses:

- Windows Azure crypto basics
- Key storage and security threats
- Immutability and in-memory resources
- Message queues

Technologies discussed:

Windows Azure

So why would I think this information is needed within the developer community? Over the last several months I've seen an increasing number of posts on the community sites regarding security in general with Azure. Microsoft has suggested encryption as part of securing application-layer data with Azure projects. However, proper understanding of both encryption and the .NET security model will be needed by product designers and developers building on the Windows Azure platform.

One thing I noticed was an increasing percentage of posts specific to crypto services and key storage. This was especially true with regards to Windows Azure Storage services. It got my own curiosity going, and I discovered it was a worthy topic to discuss in some depth.

During the course of this article, I'll be making heavy use of Cryptographic Service Providers (CSPs), which are implementations of cryptographic standards, algorithms and functions presented in a system program interface. For the purposes of this article I'll be using the symmetric encryption algorithm provided by the Rijndael cryptography class.

Crypto Basics

The Windows Azure SDK extends the core .NET libraries to allow the developer to integrate and make use of the services provided by Windows Azure. Access to the CSPs has not been

Get the most comprehensive file coverage from any of Aspose's 4 Award Winning Suites

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

•DOC	•DOCX	•WordML	•OOXML	•ODF	•RTF	•SpreadsheetML	•XLS	
•XLSX	•CSV	•TAB	•PDF	•FO	•PPT	•POT	•SWF	•& More...

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 and JasperReports. Extensitve 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™

restricted within Windows Azure projects and services. This means much of your development with regard to encrypting and decrypting data will remain the same with regards to the assemblies you're accustomed to using. However, there are changes in the underlying architecture, issues of when or where to encrypt data and where and how to persist keys. I'll discuss key and secret data persistence a little later in this article.

You also have access to the full array of cryptographic hash functionality in Windows Azure, such as MD5 and SHA. These are vital to enhance the security of any system for things such as detecting duplicate data, hash table indexes, message signatures and password verification.

A consistent recommendation is to never create your own or use a proprietary encryption algorithm. The algorithms provided in the .NET CSPs are proven, tested and have many years of exposure to back them up. Using XOR to create your own cipher process is not the same, and does not provide the same level of data security.

A second recommendation is to use the RNGCryptoServiceProvider class to generate random numbers. This ensures random numbers generated by your application will always have a very high level of entropy, making it hard to guess at the patterns.

Access to the CSPs has not been restricted within Windows Azure projects and services.

The code below implements a single static member that returns a 32-bit int value that is random and meets the requirements to be cryptographically secure. This is made possible by using the byte generator in the RNGCryptoServiceProvider found in the Cryptography namespace:

```
public static int GenerateRandomNumber() {
  byte[] GeneratedBytes = new byte[4];
  RNGCryptoServiceProvider CSP = new RNGCryptoServiceProvider();
  CSP.GetBytes(GeneratedBytes);
  return BitConverter.ToInt32(GeneratedBytes, 0);
}
```

Figure 1 shows a simple example of using the CSPs within the Windows Azure platform. Three public members are exposed for use within any Windows Azure application. The first accepts a binary key and initialization vector (IV), as well as a binary buffer of unencrypted data and returns its encrypted equivalent. The second member does the reverse by decrypting the data. The third member returns the calculated hash value for that data. Notice here that I'm using the Rijndael CSP for managed access to a provider. I'm also storing data and keys in binary buffers and writing over them as soon as I'm finished with them. I'll touch on this topic later when I discuss immutability.

This is the simplest example of encrypting data and returning the encrypted results as a byte array. This is not code that should be used in a secure environment without all the proper security analysis, only an example.

The example in **Figure 2** has an almost identical structure to the one in **Figure 1**. In this case, I'm decrypting data based on the same key and IV, only with an encrypted byte buffer as a parameter. The only real difference here is that when I create the encryption stream, I specify that I'm creating a symmetric decryptor and not an encryptor as I did previously.

Figure 1 **Simple Encryption**

```
public static byte[] SampleEncrypt(byte[] dataBuffer,
  byte[] Key, byte[] IV) {

  MemoryStream InMemory = new MemoryStream();
  Rijndael SymetricAlgorithm = Rijndael.Create();
  SymetricAlgorithm.Key = Key;
  SymetricAlgorithm.IV = IV;
  CryptoStream EncryptionStream = new CryptoStream(InMemory,
    SymetricAlgorithm.CreateEncryptor(), CryptoStreamMode.Write);
  EncryptionStream.Write(dataBuffer, 0, dataBuffer.Length);
  EncryptionStream.Close();
  byte[] ReturnBuffer = InMemory.ToArray();
  return ReturnBuffer;
}
```

Key Storage and Persistence

As with any encryption strategy at the application or enterprise layer, the encryption and decryption infrastructure is less than half the battle. The real problem comes with key storage and key persistence. The data security provided by encrypting data is only as secure as the keys used, and this problem is much more difficult than people may think at first. Systems I've reviewed have stored crypto keys everywhere, from directly in source code, to text files named something clever, to flat files stored in hard-to-find directories.

An important question of key persistence comes about when considering where to store and keep keys in a cloud environment. Some people have expressed concern that by persisting keys in the cloud you're exposing yourself to a security threat from the cloud itself. That is, if someone can get physical access to your data, data stored on disk may not be encrypted by default (as is the case with Windows Azure). Considering that SQL Azure does not yet support encryption either, this becomes a security decision to be considered in the planning and design of your solution. As with any security implementation, the risks must be measured, weighed and mitigated.

But that doesn't mean cloud platforms in general—and Windows Azure in particular—are inherently not secure. What other options may be available to you?

One thing to note right away is that no application should ever use any of the keys provided by Windows Azure as keys to encrypt data. An example would be the keys provided by Windows Azure

Figure 2 **Simple Decryption**

```
public static byte[] SampleDecrypt(byte[] dataBuffer,
  byte[] Key, byte[] IV) {

  MemoryStream InMemory = new MemoryStream();
  Rijndael SymetricAlgorithm = Rijndael.Create();
  SymetricAlgorithm.Key = Key;
  SymetricAlgorithm.IV = IV;
  CryptoStream EncryptionStream = new CryptoStream(InMemory,
    SymetricAlgorithm.CreateDecryptor(), CryptoStreamMode.Write);
  EncryptionStream.Write(dataBuffer, 0, dataBuffer.Length);
  EncryptionStream.Close();
  byte[] ReturnBuffer = InMemory.ToArray();
  return ReturnBuffer;
}
```

for the storage service. These keys are configured to allow for easy rotation for security purposes or if they are compromised for any reason. In other words, they may not be there in the future, and may be too widely distributed.

Storing your own key library within the Windows Azure Storage services is a good way to persist some secret information since you can rely on this data being secure in the multi-tenant environment and secured by your own storage keys. This is different from using storage keys as your cryptography keys. Instead, you could use the storage service keys to access a key library as you would any other stored file. This is fairly straightforward to implement. For example, say you wanted to implement your own key library as a simple text file to persist some secret information. This would be best stored as data in the blob service API as opposed to either the queue or table storage service. The blob area of the storage service is the best place for data such as binary audio and images or even text files. The queue portion of the service is focused on secure messaging for small data objects that do not persist for long periods of time. The table storage system is great for structured data and information that needs to be persisted and accessed in specific parts, identical to relational data in a database.

Windows Azure queues provide a similar set of functionality to the Microsoft Message Queuing (MSMQ) services.

You start by persisting a key in a CSP key container. This is a great option for storing a public key that is difficult to retrieve without physical access to the server. With Windows Azure, where the location of applications and data is abstracted, this would make even a public key stored in this manner extremely difficult to find and retrieve. The creation of a key storage container is very simple; here is an example using the RSA provider that creates our key. If the key container already exists, its key is loaded into the provider automatically:

```
CspParameters CspParam = new CspParameters();
CspParam.KeyContainerName = "SampleContainerName";
RSACryptoServiceProvider RSAProvider = new
  RSACryptoServiceProvider(CspParam);
```

There are also other options you can consider based on your needs. For example, you can use specific flags to secure the key to the user that created the container. This can be done with the use of CspParameters flags member:

```
CspParam.Flags = CspProviderFlags.UseUserProtectedKey;
```

Now create a request to the blob API using your Windows Azure storage key. The request itself requires both a signature string as well as a proper request header. The proper header format is:

```
Authorization="[SharedKey|SharedKeyLite] <AccountName>:<Signature>"
```

In this case, I want to maximize the security of my persisted secret data, so I'll use the SharedKey authorization method. The signature portion of the header is a hash-based authentication code that is generated by the SHA256 algorithm and your storage key against the data in the signature. This hash is then encoded into a base64 string. A sample signature might look like this:

```
"PUT\n\ntext/plain; charset=UTF-8\n\nx-ms-Date:Fri, 12 Sep 2009
22:33:41 GMT\nx-ms-meta-m1:v1\nx-ms-meta-m2:v2\n/exampleaccount/
storageclientcontainer/keys.txt"
```

As described earlier, I would then generate the base64 encoded hash and use that in the header as the signature. This key file then could only be accessed by those who have an application that runs in your application space in the Windows Azure cloud with access to your storage keys. So with key persistence, you can either manage the keys outside the Windows Azure framework or inside the cloud itself.

Key and Security Threats

One item worth covering at least briefly is key security. This is a slightly different topic than how you persist and store keys. Keys themselves are essentially strings of characters that have a very high level of entropy, meaning an extremely high level of randomness. In fact, this can lead to a common attack process to find keys within a system. For instance, if you take a dump of memory or an area of data on a hard disk, the areas of extremely high entropy are great places to start mining for keys.

Apart from choosing good security practices based on the needs of your application and securing your data, how else can you protect yourself? To start, always assume that the processes you're using to decrypt, encrypt and secure data are well-known to any attacker. With that in mind, make sure you cycle your keys on a regular basis and keep them secure. Give them only to the people who must make use of them and restrict your exposure to keys getting outside of your control.

Finally, invest time in diagramming the flow of your data, both secure and unsecure. Take a look at where your data goes and how, where you store secrets, and especially where your data crosses boundaries such as public and private networks. This will give you a good idea of where your data is exposed, and allow you to target those risks with plans for mitigating them in a straightforward manner.

A related question I've been asked is whether Windows Azure supports SSL. The short answer to this is yes! Windows Azure would not be a very capable cloud platform for Web-based services and applications without support for SSL.

Encryption with SQL Azure

The release of SQL Server 2008 introduced a new feature: transparent data encryption (TDE). For the first time, SQL Server can encrypt its data fully with very little effort needed beyond what was required for the limited encryption available in SQL Server 2005. However, the initial version of SQL Azure storage does not yet support database-level encryption, though it's a feature being considering for a future version. It should be noted that SQL Azure is only available via port 1433 and only via TCP connections; it currently cannot be exposed on other ports.

Even though this feature is not yet integrated into Windows Azure, there are several security features of SQL Azure that the developer or designer should keep in mind. First of all, SQL Azure supports the tabular data stream (TDS). This means you can for the most

INFUSE YOUR UI WITH THE POWER TO EMPOWER DECISION MAKERS

NetAdvantage for Silverlight Data Visualization is a comprehensive collection of User Interface Controls to **Build** Rich Dashboards, **Visualize** Business Data and **Empower** Decision Makers. Go to infragistics.com/sldv today to get the power of Infragistics behind you and create high end BI applications without writing a lot of code.

Infragistics Sales 800 231 8588
Infragistics Europe Sales +44 (0) 800 298 9055
Infragistics India +91-80-6785-1111

Infragistics®
KILLER APPS. NO EXCUSES.

part connect and interact with the database just like you've always done. Taking advantage of ADO.NET encryption and trusted server certificates is definitely worth considering, especially when accessing your SQL Azure database from outside the cloud.

The connection properties Encrypt=True and TrustServerCertificate = False, in the proper combination, will ensure data transmission is secure and can help prevent man-in-the-middle attacks. This is also a requirement for connecting to SQL Azure—you cannot connect to SQL Azure unless connection-level encryption has been turned on.

The data security provided by encrypting data is only as secure as the keys used.

The second security feature of SQL Azure you should familiarize yourself with is the SQL Azure firewall. This tool will be very familiar to those who have used local software firewalls or even SQL Server security surface-area toolsets. It lets you allow or prevent connections from various sources, all the way down to specific IP addresses or ranges. The SQL Azure firewall can be managed via the SQL Azure portal or directly in the master database with the provided stored procedures such as sp_set_firewall_rule and sp_delete_firewall_rule.

As with any implementation of SQL Server, user account management is another aspect that must be tightly controlled. The firewall within SQL Azure is indeed a great tool, but it should not be relied on by itself. User accounts with strong passwords and configured with specific rights should be used as well to complement your data security model.

These new tools go a long way toward making SQL Azure a very tightly secured managed platform for cloud-based applications. If you're trying this service out for the first time, remember that before you can connect, you must initially configure the SQL Azure firewall. This must first be done through the SQL Azure Web portal, but can be done later directly in the master database as described earlier.

Figure 3 **Clearing Data from Memory**

```
public static int GenerateRandomNumber() {
  byte[] GeneratedBytes = null;

  try {
    GeneratedBytes = new byte[4];
    RNGCryptoServiceProvider CSP =
      new RNGCryptoServiceProvider();
    CSP.GetBytes(GeneratedBytes);
    return BitConverter.ToInt32(GeneratedBytes, 0);
  }
  finally {
    for (int x = 0; x < GeneratedBytes.Length; x++) {
      GeneratedBytes[x] = 0;
    }
  }
}
```

Immutability and In-Memory Resources

Immuta-what? Immutability in object-oriented programming simply means that the object's state cannot be modified after its initial creation. A concrete example in the Microsoft .NET Framework is the string class. When the value of a string is changed in code, the original string in memory is simply abandoned and a new string object is created to store the new value.

Why is this important from a security perspective? Well, that string may stay in memory for as long as the server is online without a reboot. You really have no way of knowing with certainty how long a string will stay in memory. This is important when considering how to store information in code such as cryptographic keys or copies of encrypted and decrypted data. By leaving a trail of that data behind you in memory, you leave behind information that exposes your secrets to the clever data thief.

Because of this vulnerability, it is always recommended that such data be stored in buffers such as byte arrays. That way, as soon as you're done with the information, you can overwrite the buffer with zeroes or any other data that ensures the data is no longer in that memory.

Because Windows Azure is a cloud environment I've been asked if this is a still a concern, and it's a good question. True, in the Windows Azure system individual applications are isolated from each other. This makes exposing data in memory much less of an issue in general. It would be very difficult to associate applications and memory space in the cloud. However, I still recommend the cautious approach and cleaning up after yourself. You may not always run this piece of code in the cloud, and other vulnerabilities may expose themselves in the future. While less of a concern, keep this habit, and persist this approach.

In **Figure 3** I've modified the previous example that generated random integers. Here I added a bit of error-handling to ensure that I have a finally block that always runs, no matter what. Within that block I am doing a very simple iteration through the values in the byte array, overwriting each position with a zero. This overwrites that data in memory because byte arrays are mutable. I know that this number is no longer in memory owned by the execution of this member. This can be done to any byte array used as a data buffer for items such as keys, initialization vectors, and encrypted or decrypted data.

Message Queues

Windows Azure queues provide a similar set of functionality to the Microsoft Message Queuing (MSMQ) services that are common to enterprise Windows applications. The message queue service within Windows Azure stores text-based messages no larger than 8 KB in a first-in, first-out (FIFO) manner. This allows services and applications running on different servers—or in this case within the cloud—to interact and send actionable messages to each other in a secure and distributed manner.

There are five basic functions that allow you to push a message to the queue, peek a message, pull a message and so on. The question that has come up most often is, how secure are these messages?

Many of the features currently supported by MSMQ are not yet supported within the Windows Azure messaging APIs. However, there are similarities. As with the blob data service, the messaging services makes use of the same REST get and put interfaces. Writing and reading messages can be done either in code or with a URI and Web request calls that can be encrypted via SSL for requests over unsecured networks. This means transmission of requests is encrypted.

Always assume that the processes you're using to decrypt, encrypt and secure data are well-known to any attacker.

Also, as with the other storage services within Windows Azure, any access to a message queue must make use of the same storage service key. Only applications with access to the key can view or add messages to these queues. This makes any additional encryption of these messages overkill unless the body of these messages is going to be leaving the secured network or secured application space.

Wrapping It All Up

In today's drive toward service-oriented architecture and solutions, few can consider doing business without cloud applications. The isolation of data and services in a multi-tenant environment such as Windows Azure is one of the major concerns of anyone who has an eye toward using private data.

As with any new platform, security and cryptography features will continue to evolve in the Windows Azure platform. Microsoft has taken great pains to not only provide a secure, isolated environment, but also to expose what it has done to allow for public certification of these measures. This should give engineers confidence that Microsoft wants to be a closer partner on security and keeping systems and applications locked down.

The whole point of security and especially cryptography is to make your information and processes very hard to gain access to. We can define "hard" as meaning that it is beyond the capability of any adversary to break into such a system for the duration of the life of that data or process. This is, however, a relative definition based on the requirements of the application or data being used. That is why I've continued to emphasize the need of constant evaluation of security and cryptographic requirements through this article. That is essential to ensuring these tools can be used effectively to make your cloud system secure and to protect your data. ■

Jonathan Wiggs *is currently a principal engineer and development manager at Nuance Communications Inc. Read his blog at jonwiggs.com or contact Wiggs directly at Jon_Wiggs@yahoo.com.*

InishTech Software Licensing and Protection Services

LICENSE MANAGE PROTECT

Developed the Best Product? That's only Half the Battle!

Exploit your product's sales potential to the max and protect your core IP with the industry's strongest code and license protection. Make it easy for customers to buy with configurable features and flexible business terms including subscription and pay-as-you-go with InishTech's Software Licensing and Protection Online Service (SLPS).

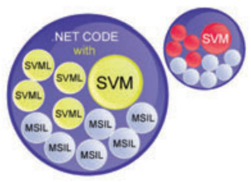
License your Products

Configure products, features and business models independent of code.

Manage your Customers

Monitor usage and manage upgrade cycles to reduce support costs and increase sales.

Protect your Hard Work



Patented Secure Virtual Machine technology protects licenses and IP with no impact on code.

SLPS is powerful, yet easy to setup and manage. Subscriptions start at just $45 per month and most customers see a return on their investment within the first six months. Contact us at sales@inishtech.com or check out our website on **www.inishtech.com/redmond** You'll be glad you did!

XCEED

DataGrid

for WPF

The most advanced WPF data presentation control

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, solid foundation, takes true advantage of WPF and provides the extensibility required for IBM U2 customers to take their applications to the next level in presentation design and styling."

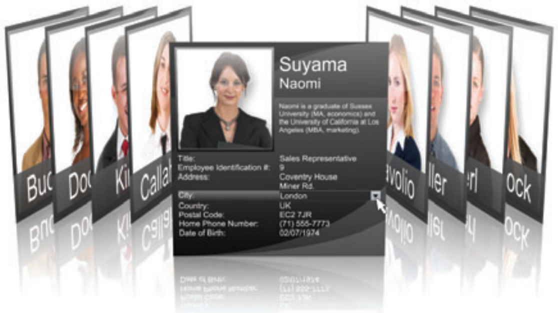
Vincent Smith
U2 Tools Product Manager at IBM

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

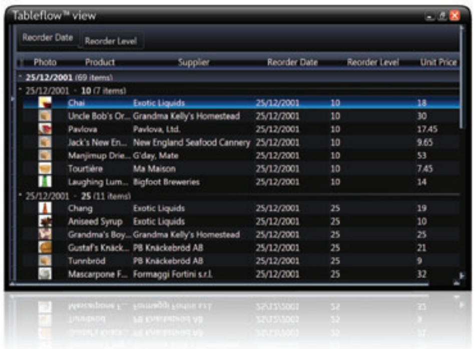


Suyama
Naomi

Naomi is a graduate of Sussex University (MA, economics) and the University of California at Los Angeles (MBA, marketing).

Title:	Sales Representative
Employee Identification #:	9
Address:	Coventry House Miner Rd.
City:	London
Country:	UK
Postal Code:	EC2 7JR
Home Phone Number:	(71) 555-7773
Date of Birth:	02/07/1974

The Coverflow™ 3D view provides a true 3D experience to add to your application. Also offered is a classic 2D card view.



Tableflow™ view

Reorder Date Reorder Level

Photo	Product	Supplier	Reorder Date	Reorder Level	Unit Price
25/12/2001 (69 items)					
25/12/2001 - **10** (7 items)					
	Chai	Exotic Liquids	25/12/2001	10	18
	Uncle Bob's Or...	Grandma Kelly's Homestead	25/12/2001	10	30
	Pavlova	Pavlova, Ltd.	25/12/2001	10	17.45
	Jack's New En...	New England Seafood Cannery	25/12/2001	10	9.65
	Manjimup Drie...	G'day, Mate	25/12/2001	10	53
	Tourtière	Ma Maison	25/12/2001	10	7.45
	Laughing Lum...	Bigfoot Breweries	25/12/2001	10	14
25/12/2001 - **25** (11 items)					
	Chang	Exotic Liquids	25/12/2001	25	19
	Aniseed Syrup	Exotic Liquids	25/12/2001	25	10
	Grandma's Boy...	Grandma Kelly's Homestead	25/12/2001	25	25
	Gustaf's Knäck...	PB Knäckebröd AB	25/12/2001	25	21
	Tunnbröd	PB Knäckebröd AB	25/12/2001	25	9
	Mascarpone F...	Formaggi Fortini s.r.l.	25/12/2001	25	32

The new smooth-scrolling Tableflow™ view provides an amazingly rich, fluid, and high-performance user experience.

Try it live on xceed.com

XCEED
MULTI-TALENTED COMPONENTS

Surprise!

9 Useful Tactics for Paying Back Technical Debt

David Laribee

In the December 2009 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee819135) I gave advice for identifying and building a case to tackle technical debt. In summary, I believe it's important to identify the debt that's likely to harm you in the near future. Introducing technical excellence to seldom touched parts of your codebase won't help you realize productivity gains tomorrow.

Also, I hope that you understand the importance of obtaining license and buy-in from management on the importance of paying back debt and have some basic tools to start building a rock-solid case for the same.

Now let's turn our attention to tactics that might help you pay back high interest technical debt. There are many proven tactics in dealing with technical debt. A full catalog of the patterns, tools and techniques for wrangling difficult code is well beyond the scope of this article. Instead, I'll supply some of the more applicable tricks I've added to my repertoire over the years.

This article discusses:

- Continuous learning
- Working together effectively
- Test and measure results
- Move forward, but don't stop improving

Technologies discussed:

Testing, Design Patterns

Learn, Learn, Learn

If you know you have issues, but you're not sure how to fix them, it might be time to acquire new knowledge and skills that will help you raise your code out of the muck. Learning, as they say, is fundamental.

Learning can take many forms. You might need outside help in the form of consultants or classroom training. You might be able to get by with books.

Try to involve your team in the learning process. Perhaps you could start a book club within your team. Maybe you can bring back the benefits of a course or conference in the form of an instructive presentation.

A collaborative and hands-on technique for involving the whole team is the Coding Dojo. A basic Coding Dojo involves picking a programming challenge and tackling that as a group. I've experimented with a rotating pair watched by a peanut gallery. In this method, two members of the team work together on a programming task, with "tag" intervals where other members of the team enter the dojo as another person leaves.

If you learn best at your own pace or want to start a book club, there are a couple of good texts I can recommend on the subject of improving the maintainability of legacy code.

Michael Feathers' aptly titled volume, "Working Effectively with Legacy Code" (Prentice Hall 2004), provides a patterns-based approach to teasing out legacy code. The author makes the statement that legacy code is untested code. It is difficult to change, and you can't be certain your changes aren't introducing regression defects. In this book you'll find a number of focused strategies and tactics for reducing coupling in your code and making it more testable.

Kyle Baley and Donald Belcham have one of the newest books on the scene, "Brownfield Application Development in .NET" (Manning Publications, 2010). They take a systemic approach toward improving so-called brownfield (versus new development. or greenfield) codebases. One benefit of this book is that, while the approaches they recommend are broadly applicable, their code examples are designed around the Microsoft .NET Framework, a likely benefit to readers of this article. I quite like how they take a team-practices approach as well. That is, while you're making changes in a wild codebase, the confidence you'll get from implementing some basics such as continuous integration and version control is worth its weight in gold.

There's a high probability the messy code you have to deal with was written by someone currently on your team.

Diplomacy

There's a high probability the messy code you have to deal with was written by someone currently on your team. It's important that you take this into account when reasoning about the code in its current state. Hurt feelings lead to defensiveness that, in turn, leads to slow going on the improvement train.

Try defusing the situation with anecdotes of mistakes you've made in the past. Stay professional, avoid personal attacks and encourage the author of the original code to make suggestions about how you might go about improving it.

Then again, it's quite possible you're one of the developers that contributed to your mess. I want you to repeat after me: "I am not my code. I am learning every day and am dedicated to finding a better way moving forward. I will not let my colleagues' critiques or my own ego stand in the way of my team's effort to improve."

In truth, it takes time to get over these issues. I find the best way to reason and talk about improvements is to focus on the present and near future rather than the past. What could this code be? What do you want to see it evolve into?

A little diplomacy and consideration for other people's emotional investment in work that's already been committed will go a long, long way toward moving forward.

Introduce a Shape

Some code is so horrendous it's hard to understand what's going on at all. Perhaps all classes are in a single namespace. Perhaps the codebase is in such a tangled web of dependencies that following the stack greatly exceeds your short-term memory's ability to keep your place.

Symptoms like these often imply a diagnosis of debt at the architectural and design levels rather than at an implementation level. This, as far as I'm concerned, is the most insidious kind of debt and usually leads to the greatest costs of change.

Brian Foote and Joseph Yoder call architectures with no discernible shape, where everything depends on everything else, the "big ball of mud" (laputan.org/mud):

"A big ball of mud is a casually, even haphazardly, structured system. Its organization, if one can call it that, is dictated more by expediency than design. Yet, its enduring popularity cannot merely be indicative of a general disregard for architecture."

I'd bet my last dollar that most software applications in production today are big balls of mud. This isn't necessarily a value judgement. There are billions of lines of terrible code out there in the world making people lots and lots of money. It stands to reason that big balls of mud are fulfilling the champagne dreams and caviar wishes of many a business owner and shareholder.

The problem is that ball-of-mud applications become increasingly costly to change. While the business environment remains dynamic, the software becomes inflexible. The typical strategy for dealing with this is the software equivalent of a nuclear bomb: the big rewrite. There are many risks associated with big rewrites and it's often better to try to improve the design of the incumbent system.

Before you can start to employ some of the lower-level techniques, it's often valuable to introduce a shape to your system. The typical example is that of a layered architecture. Classically this means the UI talks to services and services talk to some kind of model and the model, in turn, talks to your persistence layer.

Shaping up your code into layers can be a very low-fidelity activity. Start by organizing code into namespaces named after the layers of your architecture.

Now you have your marching orders: enforce the rule that higher-level layers (user interface layer) may only depend on the next level up (services layer). As simple way of enforcing the rule is to move your layers into separate projects in Visual Studio. The solution won't compile if you violate the rule.

When you cover your code with tests before you change the code, you're more likely to catch any mistakes.

By making the rule pass, you have decreased coupling. The model is no longer coupled to your application's views. By introducing a shape you have increased cohesion. Classes inside your layer are all working to the same purpose whether that be to display data to an end user or to encapsulate business behavior.

Introduce facades between layers and make higher level layers such as your UI depend on facades provided by lower level layers rather than granular classes inside the layers. You can apply this technique this process incrementally and opportunistically.

Figure 1 **NDepend CQL**

```
-- Efferent coupling outside a namespace
SELECT TYPES
WHERE TypeCe > 0
      AND (FullNameLike "MyCompany.MyProduct.Web")

-- Afferent coupling inside a namespace
SELECT TYPES
WHERE TypeCa > 0
      AND (FullNameLike "MyCompany.MyProduct.Web")

-- Top 20 most complicated methods
SELECT TOP 20 METHODS
WHERE CyclomaticComplexity > 4
      AND FullNameLike "MyCompany.MyProduct.Web"
```

The power of a imposing a shape on the monolithic big ball of mud is you can now start to identify more targeted opportunities for paying back technical debt. That is, if you're doing lots of work in, say, CompanyX.ProductY.Model, you might drill down with a static analysis tool to find the most coupled or complicated classes.

Close Air Support with Tests

The process of making changes without changing system behavior is called refactoring. There are entire refactoring pattern languages dedicated for both object-oriented (refactoring.com) and relational-database (agiledata.org/essays/databaseRefactoringCatalog.html) code: Extract Method, Split Table and so on. The fact of the matter is, it's difficult to apply these granular and safe methods when you don't fully understand the code base.

So how do you start making changes in a legacy project? The first thing to notice is that, given a choice, it is always safer to have tests around the changes that you make. When you change code, you can introduce errors. But when you cover your code with tests before you change the code, you're more likely to catch any mistakes.

The practice of shotgun surgery, plunging headlong into code without any real confidence the changes you're introducing aren't also introducing dangerous defects, isn't the only way to force a change.

Before you start changing code, determine whether there's a hard interface in the system against which you can write tests. These tests are of the black-box variety. That is, you're feeding a system inputs and inspecting the outputs. When you make your change, continually run the tests to verify your changes haven't broken existing behavior.

Application of this tactic can be challenging when you're tackling parts of your system that are tightly coupled. The cost of testing may very well exceed the benefit of removing debt. This constant cost-benefit analysis permeates the process of turning a codebase around and, sometimes, it's more cost-effective to straight up re-write an application or large section of an application's codebase.

Measure Observable Effects

Build measurements around the area of code you're improving. For the sake of argument, let's say that you're trying to better organize the core business logic of your application. There are lots of paths through the members in the types of this namespace: switch statements, nested if statements and the like. A measurement such as cyclomatic complexity can give you a rough sense of whether improvement efforts are simplifying your code.

You can obtain extremely specific measurements of specific parts of your codebase with the NDepend code-analysis tool (ndepend.com). NDepend provides a powerful Code Query Language (CQL) over namespaces, types and members in your .NET assemblies.

Consider the CQL statements in **Figure 1**. Note that I'm probing measurements like coupling and complexity (just a few of the many metrics NDepend makes available) inside a particular namespace. This implies that I've already introduced a shape so I can focus efforts in definable areas of my code. If I'm successful in introducing positive changes, I should see measurements like coupling and complexity decrease over time.

A nice side-effect of this tactic is that the measurements can help you hold the line and maintain discipline once debt has been removed. They will give you an early warning system toward the reintroduction of new debt into an already improved area.

Dedicated Improvement Stream

You don't live in a vacuum. Chances are, during your improvement efforts, you'll be asked to continue to deliver new features and modifications to existing features. Delivery pressure causes feelings of being under the gun. But maintenance is a fact of life you should embrace rather than trying to ignore.

One way to deal with this is to secure approval from the business to dedicate resources—an individual, a pair or an entire team—to improving debt items concurrently with delivering new features.

This can be a highly effective strategy but is best when the entire team (all the developers and testers who make modifications to the codebase) takes a part in the improvements being made. Try regularly rotating individuals as pairs. The developer that's been in the improvement stream the longest rotates out, leaving the other developer to brief the new pair on what's happening.

By spreading the knowledge you get closer to collective ownership, thereby reducing risk and improving designs.

Sometimes you'll find opportunities for improvement that lie directly in the way of some new functionality you're trying to deliver. Whenever you start work on a new or modified feature, it's good practice to review the list to determine whether the team hasn't already identified an area for improvement that intersects with the work you're about to do.

Maintenance is a fact of life you should embrace rather than trying to ignore.

Opportunities for improvement occur all the time, often identified on-the-fly and achieved with a few simple refactorings that make a difference the next time a teammate encounters the code.

There's a constant cost-benefit analysis that goes on when you're improving the existing code base while delivering new features. If the improvement seems too costly, add it back to your list and discuss it in your improvement planning.

Iterate, Iterate, Iterate

You've payed back some debt. Time to go back to step one and identify, prioritize and build consensus on the next item that needs fixing, right?

Yes, but there's a bit more to it than mindlessly plowing through your list. You have to be sure that you're not incurring more debt than your fixing. You should also regularly incorporate your learnings into future efforts in new development and improvement efforts alike.

Opportunities to improve a codebase change regularly. They emerge and their importance ebbs and flows. Reasons for the dynamic nature of high-interest debt change from release to release.

What's worked well for me is scheduling a short, weekly meeting with developers to review new debt items and prioritize the backlog of existing debt items. This keeps the consensus you've built alive and the list fresh. Again, I'd give priority to fixing the debt that's likely to slow down your current release or project.

Begin the meeting by reviewing new items. Have the identifier pitch their case and put it to vote: does it merit inclusion in the backlog or not? Once you've gone through the new items, review the old items. Is there work that no longer applies? Will there be immediate value in completing this work, that is, will it remove day-to-day impediments? Last, prioritize the opportunity against others—re-rank your list. The top item on the list should be the very next improvement to make.

Hold the Line

While you and your team are merrily paying down high-interest technical debt, you'll likely also be delivering new software. As you learn about solid programming techniques and introduce new patterns into your code, apply this knowledge going forward. It's possible that additive work will pile on existing technical debt creating an inescapable inertia.

It's important that you set expectations for your business stakeholder for new work. Higher quality takes more time to attain than rushed, get-it-done-style code. This fact brings me back to the systems-thinking concept introduced back in my December 2009 article. For me this is a cultural attribute. That is, organizations can either think sustainably for the long term or continue with a buy now, pay later mentality—the oh so fertile breeding ground of technical debt. Never forget the central question, how did we end up here in the first place?

While you're learning about how to improve a codebase, you will very likely develop some team norms that apply to new code. I suggest capturing these in a tool like a wiki and holding small, informal learning sessions where you share your findings with your team. You will also develop techniques for dealing with similar improvement items. When you notice you've done the same thing to correct a flawed design or clean up implementation three or four times, codify it in your team's doctrine. That is, write it down in a well-known place and, very simply, tell people it's there.

Work Together

Technical debt is a people problem. People, through lack of knowledge or unrealistic expectations, created the mess and are now dealing with the consequences. And it'll take people working as a group to fix it.

Giving advice like this is all well and good, and I'd be surprised if you, a software professional and likely one who's passionate about their craft, weren't in complete agreement.

Higher quality takes more time to attain than rushed, get-it-done-style code.

A successful turnaround requires fundamental changes in the value system of everyone involved—the entire team. The economics of quality are proven to pay back in the end, but you'll have to take that step of faith in the near term. You have to win hearts and minds in order to change a culture, and that can be a tough job indeed. The most useful suggestion I can make is: *don't go it alone*. Get the team behind the effort and make sure everyone has a stake in the results.

Setting a goal like "we want 90 percent coverage" or "we want to do Test-Driven Development (TDD) all the time" is relatively meaningless. Tackle the problem areas that are slowing you down at the moment and in the near future. That might mean introducing TDD and living by the coverage report—or it might not. It might be something more primitive like making sure your team knows the basics of object-oriented analysis and design.

Start Making a Difference

While I hope I've given you some tools and techniques for tackling debt or, at the very least, made some of the implicit ideas and experiences you've had explicit, it's important to realize that dealing with technical debt is very much a product-to-product issue. You may, for example, be in an environment where there's not a lot of trust between the development and business parties and find you have to pitch your case with the preparation of a trial attorney.

There's no out-of-box process that'll tell you the how to for driving down debt, but as for the *when* and the *where*, today is a fine day to start making a difference. March toward technical excellence can be slow and rough going in the beginning. It's only through sustained effort, constant learning and, above all, an earnest attitude that you'll pull through the tough times, bringing debt-crippled code back into the black. I encourage you to stick with the program. Not only will you increase value for your customers, you will greatly expand your craftsman's toolbox. ■

DAVE LARIBEE *coaches the product development team at VersionOne Inc. He's a frequent speaker at local and national developer events and was awarded a Microsoft Architecture MVP for 2007 and 2008. He writes on the CodeBetter blog network at thebeelog.com.*

THANKS to the following technical expert for reviewing this article: *Donald Belcham*

Text Template Transformation Toolkit and ASP.NET MVC

Microsoft Visual Studio includes a code generation engine known as T4 (which is short for Text Template Transformation Toolkit). You've probably already used T4 templates in Visual Studio without even knowing they were working behind the scenes. In this article I'm going to give you a basic introduction to T4 templates and show you how ASP.NET MVC uses this technology. I'll also show you how to customize T4 templates to enhance your day-to-day work with the MVC framework.

The basic idea behind the template toolkit is to parse an input file and transform it into an output file. The input file is a template—a text file with a .tt file extension. The output file will also contain text, and the text can be C# code, Visual Basic code, Web Forms code, markup or anything else you need to generate.

The easiest way to see T4 in action is to create a new project in Visual Studio. I'll be generating C# code in this article, so you can use any project type that compiles C# code. Once the project is opened, right-click the project and select Add | New Item. Select Text File from the Add New Item dialog (there's no item template dedicated to T4 in Visual Studio 2008, but there will be in 2010), and name the file Simple.tt (make sure you use the .tt extension). Once the file is loaded into the project you'll immediately see a Simple.cs file appear behind Simple.tt in the Solution Explorer window (see **Figure 1**).

Both Simple.tt and Simple.cs will start as empty files. If you right-click the Simple.tt file and select Properties, you'll see that Visual Studio assigned TextTemplatingFileGenerator as the custom tool for the file (see **Figure 2**). This generator is the T4 engine that will transform the template file into a file full of C# code.

To make the template do something interesting, add the following code:

```
<#@ template language="c#v3.5" #>
<#@ assembly name="System.Web.Mvc.DLL" #>
<#@ import namespace="System.Web.Mvc" #>

public class Test
{
<# for(int i = 0; i < 5; i++) { #>
  public int Prop<#= i #> { get; set; }
<# } #>
}
```

The code begins with some directives. Directives allow you to specify the programming language for the template and include namespaces and assemblies required by the code in the *template*. I want to stress that I'm talking about settings required to execute code in the template and not code in the project itself. You can also use a directive to specify the extension of the output file. The default is C#, but as I mentioned earlier, you can generate Visual Basic code, XML, HTML or any other textual artifact.

The directives I'm using tell the template engine to use the C# compiler that comes with the Microsoft .NET Framework 3.5. It also tells the template engine to reference the ASP.NET MVC assembly and to bring the System.Web.Mvc namespace into scope. The MVC assembly and namespace are not actually required by the simple code in the template, but I put them in the template as an example.

After the directives, the text you see that's not between <# and #> delimiters is put verbatim into the output file. The text between <# and #> is C# code. The template engine will parse the code and add it to a class for execution (a class ultimately derived from the TextTransformation class in the Microsoft.VisualStudio.TextTemplating assembly). This process is similar to the ASP.NET view engine process where the code and markup in an .aspx file are added to a class ultimately derived from System.Web.UI.Page. If you've already been writing your MVC views using the Web Forms view engine, you'll feel comfortable creating templates. In .aspx files you can use C# code to generate HTML. In my .tt file, I'm using C# code to generate C# code.

The code I have in Simple.tt will produce the following C# output in Simple.tt.cs:

```
public class Test
{
  public int Prop0 { get; set; }
  public int Prop1 { get; set; }
  public int Prop2 { get; set; }
  public int Prop3 { get; set; }
  public int Prop4 { get; set; }
}
```

Of course, the Test class is completely useless and wholly uninteresting, but I hope it gives you some idea of the possibilities that exist with T4 templates. Because you're writing C# code in the template, you can connect to databases, read data from the file system,

T4 Editing

When you're editing T4 templates in Visual Studio, you'll have no help from the language services in the IDE, like IntelliSense and syntax highlighting. There are two solutions to this problem. One is the Visual T4 editor available from Clarius Consulting (visualt4.com). Another solution is the Tangible T4 Editor from Tangible Engineering (t4-editor.tangible-engineering.com).

Send your questions and comments to extrmasp@microsoft.com.

parse XML or use any .NET class to connect and read metadata that exists somewhere in your environment. This metadata, like a database schema or the types inside another assembly, is information you can use to generate classes. The classes will become part of the current project, so they will compile into the current assembly and you can use them in your application.

With a basic understanding of how T4 templates work, let's look at how the MVC framework uses T4 templates.

T4 in ASP.NET MVC

You've been using T4 templates every time you used the Add View or Add Controller features in an ASP.NET MVC project. These templates are located in your Visual Studio installation within the Common7\IDE\ItemTemplates\CSharp\Web\MVC 2\CodeTemplates folder. Visual Basic versions of the template also exist, but I'll leave it as an exercise for the reader to deduce the folder name.

The templates themselves provide a great education on the value and features of T4. For example, here is an excerpt from List.tt in the AddView subfolder of CodeTemplates:

```
if(!String.IsNullOrEmpty(mvcViewDataTypeGenericString)) {
  Dictionary<string, string> properties =
    new Dictionary<string, string>();
  FilterProperties(mvcHost.ViewDataType, properties);
#>
  <table>
    <tr>
      <th></th>
<#
  foreach(KeyValuePair<string, string> property in properties) {
#>
      <th>
        <#= property.Key #>
      </th>
<#
  }
#>
```

The job of List.tt is to produce an .aspx file that will display a collection of model objects in tabular form. In the template you can see the table, tr and th tags being written. To produce the .aspx file the template needs some contextual information, like the name of the master page it should use and the type of the model. The template can retrieve this information from its host object. The host object sits between a template and the T4 engine and can give a template access to resources (like local files) and environmental settings. Typically, the host is Visual Studio, but the MVC team created a custom host of type MvcTextTemplateHost in the Microsoft.VisualStudio.Web.Extensions assembly. It's this custom host object that carries forward information you enter in the Add View and Add Controller dialog boxes, which are the closest things you'll find to wizards in an MVC project.

List.tt will loop through the displayable properties of the strongly typed model object and create a table with a column for each property. The template uses reflection to discover the model's available properties in a FilterProperties method. FilterProperties is a helper method defined later in the template file. The template also sets up links to navigate to the edit and details actions, and sets up the proper @ Page or @ Control directives for the .aspx, depending on whether you're creating a view or a partial view.

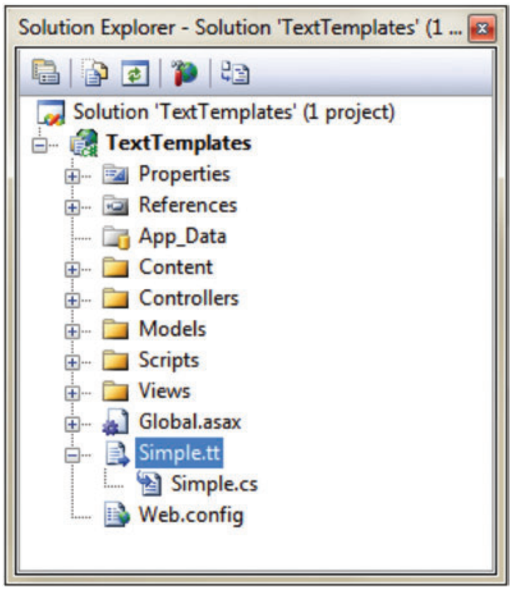


Figure 1 **C# File Behind a T4 Template**

When the template is finished running, you'll have a new .aspx view with everything you need to display a collection of model objects. Chances are you'll go into the .aspx file and perform some fine-tuning to make the view consistent with the look and feel of the views in the rest of your application.

If you find you're always making the same changes to these generated views (or to the controller code generated by Controller.tt), you can save time by modifying the templates themselves. For example, you could modify the built-in templates to add class attributes for style rules you use in your project, or perhaps something even more drastic. Keep in mind that modifying the template files in the Visual Studio installation directory will change the code generated in all the projects you work with on your machine. If you want to change the generated code for a single project, you can do this, too.

Per-Project T4 Customization

If you want custom versions of the code-generation templates on a per project basis, your first step is to copy the CodeTemplates folder from the Visual Studio installation and paste it into the root of your ASP.NET MVC project. You don't need to copy all the templates into your project, however. You can copy only the templates you want to modify. There are a total of six MVC code-generation templates, one for adding a controller (Controller.tt) and five for adding views (Create.tt, Details.tt, Edit.tt, Empty.tt, List.tt). If a template exists in your project, it will override the template in the Visual Studio installation directory.

When you add a .tt file to a Visual Studio solution, the IDE will automatically assign the .tt file a custom tool setting of TextTemplatingFileGenerator. You've already seen this behavior if you created the Simple.tt template I discussed earlier. However, this is

MvcTextTemplateHost Properties

Note that not all of the properties on the MvcTextTemplateHost object are available for every context. The templates execute when you invoke the Add View and Add Controller context menu items. The Namespace property is available for both these operations and will be set to the appropriate value. The MasterPage property, however, is only set to a valid value during an Add View operation and will contain the value the user entered for the MasterPage name in the Add View dialog.

not the proper setting for the MVC T4 Templates. The MVC tools for Visual Studio will invoke these templates at the appropriate times and create the special MvcTextTemplateHost object during template processing. Thus, after copying the templates into your project, the second step is to open the Properties Window for each template file and remove the Custom Tool setting (leave the setting blank). At this point you're ready to customize your templates.

As an example, let's say you do not want your controllers to have an Index action. You'd rather use a default action named List. You can open up the Controller.tt template in the CodeTemplates\AddController folder and change the appropriate area of code to look like the following:

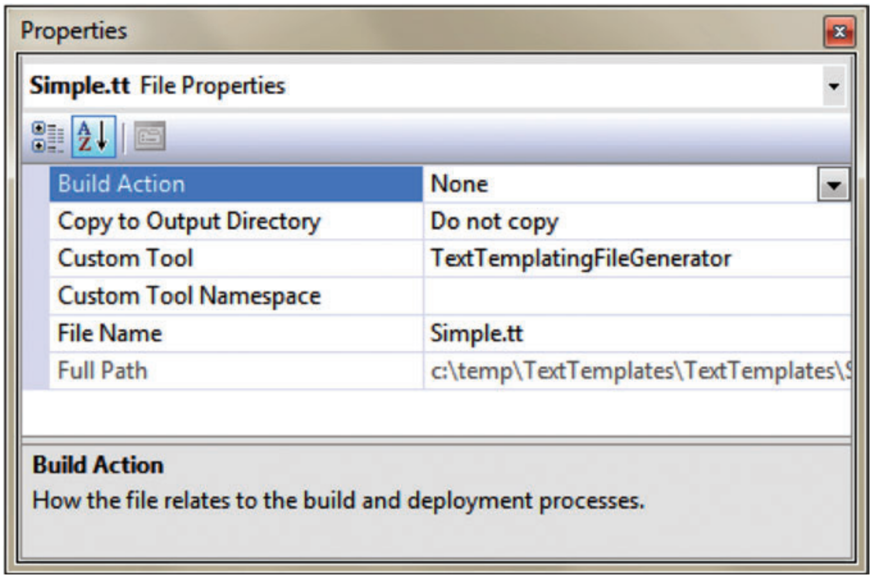
```
public class <#= mvcHost.ControllerName #> : Controller
{
  // GET: /<#= mvcHost.ControllerRootName #>/

  public ActionResult List()
  {
    return View();
  }
...
```

This is a simple change to make, but it can save you and your team quite a bit of time over the life of a large project.

One Step Further—T4MVC

In the summer of 2009, David Ebbo of the ASP.NET team created T4MVC, a T4 template designed to generate strongly typed helpers in an ASP.NET MVC application. Ebbo has refined the template over time and you can now download it from aspnet.codeplex.com/wikipage?title=T4mvc.

The T4MVC template is a traditional T4 template. You add T4MVC.tt and its associated settings file (T4MVC.settings.t4) to your project and it will use the TextTemplatingFileGenerator custom tool to generate C# code. T4MVC will help you eliminate many of the magic string literals from your MVC application. For example, one of the jobs the template will do is to scan the Content and Scripts folders in your project and generate classes with static properties to represent each script and piece of content.


Properties

Simple.tt File Properties

Build Action	None
Copy to Output Directory	Do not copy
Custom Tool	TextTemplatingFileGenerator
Custom Tool Namespace	
File Name	Simple.tt
Full Path	c:\temp\TextTemplates\TextTemplates\S

Build Action
How the file relates to the build and deployment processes.

Figure 2 **Properties of the T4 Template**

The generated code means you can render the LogOnUserControl partial view provided by the default MVC project with this code:

```
<% Html.RenderPartial(MVC.Shared.Views.LogOnUserControl); %>
```

Previously you would have used a string literal:

```
<% Html.RenderPartial("LogOnUserControl"); %>
```

If someone renames, moves, or deletes the LogonUserControl, the strongly typed code will produce a compilation error when the view compiles. In addition to strongly typed access to views and partial views, the T4MVC template also provides strongly typed access to all files inside your Content and Scripts folders and strongly typed access to controllers and controller actions.

T4MVC is a T4 template designed to generate strongly typed helpers in an ASP.NET MVC application.

You can use the T4MVC-generated classes when building action links, returning view results, and even when building the routing table for an application. Note that when you first add T4MVC to your project, you'll see some warnings generated in the IDE's Error List window. These warnings are just T4MVC telling you about some changes it is applying to your code. Most of these changes are nothing that will change the behavior of your application; the T4MVC templates just add some partial keywords to controller class definitions and will also make non-virtual action methods virtual. For more information on T4MVC, check out Ebbo's blog at blogs.msdn.com/davidebb.

Easier to Maintain

T4 is a wonderful treasure inside of Visual Studio but is still not well-publicized. This article gives you everything you need to get started with custom templates for your ASP.NET MVC project. Hopefully, you can find some uses for T4 Templates outside of your Web application project, too. You should also try out the T4MVC templates in your project, as they make your code easier to maintain and refactor. Moving forward, T4 technology is even better in Visual Studio 2010 with the addition of dedicated item templates and pre-compiled templates. ■

K. Scott Allen *is the principal consultant and founder of OdeToCode, and also a member of the Pluralsight technical staff. You can reach Allen through his blog (odetocode.com/blogs/scott) or on Twitter (twitter.com/OdeToCode).*

Thanks to the following technical expert for reviewing this article: *David Ebbo*

1105 MEDIA INC

MARK YOUR CALENDARS!
COMING IN AUGUST 2010...

VSLive!

Empowering Developers Since 1993

REDMOND, WA | MICROSOFT CAMPUS

DETAILS AND REGISTRATION AT

VSLIVE.COM

USE PRIORITY CODE VSLMS1

VSLive!

Empowering Developers Since 1993

THE EVENT THAT'S BEEN EMPOWERING DEVELOPERS SINCE 1993...

JOIN YOUR FELLOW DEVELOPERS FOR TOPICS ON:

- **Silverlight/WPF**
  Silverlight, WPF, XAML, Visual Studio 2010's XAML Designer, Expression Blend, WCF RIA Services
- **Web**
  Web Forms, ASP.NET MVC, ASP.NET AJAX and jQuery
- **Visual Studio 2010/.NET 4**
  Visual Studio features, TFS, new language features, parallel programming extensions
- **SharePoint**
  SharePoint 2010, Office 2010, Visual Studio Tools for Office
- **Cloud Computing**
  Includes cloud, server and messaging technologies,Windows Azure, AppFabric (for Windows Server and Windows Azure), REST services programming, Project "Dallas", WCF, Windows Workflow
- **Data Management**
  SQL Server, Microsoft BI, Entity Framework, ADO.NET Data Services, oData, Sync Services

1105 ENTERPRISE COMPUTINGGROUP   Visual Studio MAGAZINE ENTERPRISE SOLUTIONS FOR .NET DEVELOPMENT

AUGUST 2-6, 2010

REDMOND, WA
MICROSOFT CAMPUS

...ARRIVES AT THE MICROSOFT CAMPUS IN REDMOND, WA!

DETAILS AND REGISTRATION AT

VSLIVE.COM

USE PRIORITY CODE VSLMS1

Take your code to the next level at VSLive! at the Microsoft Campus in Redmond, WA.

Our goal at VSLive! is to first help you learn what you need to know about the Microsoft Development Platform, and then burrow deep into the subjects you need to master.

TED NEWARD

ACID Transactions with STM.NET

While this column has focused specifically on programming languages, it's interesting to note how language ideas can sometimes bleed over into other languages without directly modifying them.

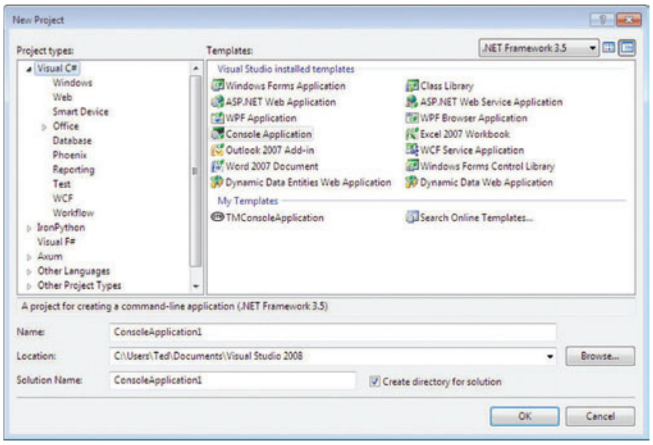
One such example is the Microsoft Research language C-Omega (sometimes written Cw, since the Greek omega symbol looks a lot like a lower-case w on the US keyboard layout). In addition to introducing a number of data- and code-unifying concepts that would eventually make their way into the C# and Visual Basic languages as LINQ, C-Omega also offered up a new means of concurrency called chords that later made it into a library known as Joins. While Joins hasn't, as of this writing, made it into a product (yet), the fact that the whole chords concept of concurrency could be provided via a library means that any run-of-the-mill C# or Visual Basic (or other .NET language) program could make use of it.

Another such effort is the Code Contracts facility, available from the Microsoft DevLabs Web site (msdn.microsoft.com/devlabs) and discussed in the August 2009 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee236408). Design-by-contract is a language feature that was prominent in languages like Eiffel, and originally came to .NET through the Microsoft Research language Spec#. Similar kinds of contractual guarantee systems have come through Microsoft Research as well, including one of my favorites, Fugue, which made use of custom attributes and static analysis to provide correctness-checking of client code.

Once again, although Code Contracts hasn't shipped as a formal product or with a license that permits its use in production software, the fact that it exists as a library rather than as a standalone language implies two things. First, that it could (in theory) be written as a library by any .NET developer sufficiently determined to have similar kinds of functionality. And second, that (assuming it does ship) said functionality could be available across a variety of languages, including C# and Visual Basic.

If you're sensing a theme, you're right. This month I want to focus on yet another recently announced library that comes from the polyglot language world: software transactional memory, or STM. The STM.NET library is available for download via the DevLabs Web site, but in stark contrast to some of the other implementations I've mentioned, it's not a standalone library that gets linked into your program or that runs as a static analysis tool—it's a replacement and supplement to the .NET Base Class Library as a whole, among other things.

Note, however, that the current implementation of STM.NET is not very compatible with current Visual Studio 2010 betas, so the usual disclaimers about installing unfinished/beta/CTP software on machines you care about applies doubly so in this case. It should install side-by-side with Visual Studio 2008, but I still wouldn't put it on your work machine. Here's another case where Virtual PC is your very good friend.



A project for creating a command-line application (.NET Framework 3.5)

Name: ConsoleApplication1
Location: C:\Users\Ted\Documents\Visual Studio 2008
Solution Name: ConsoleApplication1
[✓] Create directory for solution

Figure 1 **Starting a New Project with the TMConsoleApplication Template**

Beginnings

The linguistic background of STM.NET comes from a number of different places, but the conceptual idea of STM is remarkably straightforward and familiar: rather than forcing developers to focus on the means of making things concurrent (focusing on locks and such), allow them to mark which parts of the code should execute under certain concurrency-friendly characteristics, and let the language tool (compiler or interpreter) manage the locks as necessary. In other words, just as database admins and users do, let the programmer mark the code with ACID-style transactional semantics, and leave the grunt work of managing locks to the underlying environment.

While the STM.NET bits may appear to be just another attempt at managing concurrency, the STM effort represents something deeper than that—it seeks to bring all four qualities of the database ACID transaction to the in-memory programming model. In addition to managing the locks on the programmer's behalf, the STM model also provides atomicity, consistency, isolation and

This article is based on pre-release versions of the Microsoft .NET Framework 4 beta 1 STM.NET library.

Send your questions and comments for Neward to polyglot@microsoft.com.

RadControls *for*

Silverlight 3

Now with 35+ advanced UI controls, full .NET RIA Support, VS Extensions, free Testing tool, 5x better performance

UI COMPONENTS	DATA	PRODUCTIVITY TOOLS	AUTOMATED TESTING	TFS TOOLS	CMS
ASP.NET AJAX	OpenAccess ORM	JustCode	Web Testing Tools	Work Item Manager	Sitefinity
Silverlight	Reporting			Project Dashboard	
ASP.NET MVC					
WinForms					
WPF					

telerik

deliver more than expected

www.telerik.com

Europe HQ: +359.2.80.99.850 • US Sales: +1.888.365.2779 • Germany Sales: +49.89.8780687.70 e-mail: sales@telerik.com durability, which of themselves can make programming much simpler, regardless of the presence of multiple threads of execution.

As an example, consider this (admittedly wildly overused) pseudocode example:

```
BankTransfer(Account from, Account to, int amount) {
  from.Debit(amount);
  to.Credit(amount);
}
```

What happens if the Credit fails and throws an exception? Clearly the user will not be happy if the debit to the from account still remains on the record when the credit to the account isn't there, which means now the developer has some additional work to do:

```
BankTransfer(Account from, Account to, int amount) {
  int originalFromAmount = from.Amount;
  int originalToAmount = to.Amount;
  try {
    from.Debit(amount);
    to.Credit(amount);
  }
  catch (Exception x) {
    from.Amount = originalFromAmount;
    to.Amount = originalToAmount;
  }
}
```

This would seem, at first blush, to be overkill. Remember, however, that depending on the exact implementation of the Debit and Credit methods, exceptions can be thrown before the Debit operation completes or after the Credit operation completes (but doesn't finish). That means the BankTransfer method must ensure that all data referenced and used in this operation goes back to exactly the state it was in when the operation began.

If this BankTransfer gets at all more complicated—operating on three or four data items at once, for example—the recovery code in the catch block is going to get really ugly, really quickly. And this pattern shows up far more often than I'd like to admit.

Another point worth noting is isolation. In the original code, another thread could read an incorrect balance while it was executing and corrupt at least one of the accounts. Further, if you simply slapped a lock around it, you could deadlock if the from/to pairs were not always ordered. STM just takes care of that for you without using locks.

If, instead, the language offered some kind of transactional operation, such as an atomic keyword that handled the locking and failure/rollback logic under the hood, just as BEGIN TRANSACTION/COMMIT does for a database, coding the BankTransfer example becomes as simple as this:

```
BankTransfer(Account from, Account to, int amount) {
  atomic {
    from.Debit(amount);
    to.Credit(amount);
  }
}
```

You have to admit, this is a lot less to worry about.

The STM.NET approach, however, being library based, isn't going to get quite this far since the C# language doesn't allow quite that degree of syntactic flexibility. Instead, you're going to be working with something along the lines of:

```
public static void Transfer(
  BankAccount from, BankAccount to, int amount) {
  Atomic.Do(() => {
    // Be optimistic, credit the beneficiary first
    to.ModifyBalance(amount);
    from.ModifyBalance(-amount);
  });
}
```

Insights: It's Really a Language Change

In the course of reviewing Neward's column, one thing jumped out at me as being, unfortunately, a basic misinterpretation. Neward tries to divide language extensions into those that require language changes and those that are (purely) library changes. It tries to classify STM.NET as the latter—a library-only change—whereas I would argue it most decidedly is not.

A library-only extension is one that is implementable completely in the existing language. Library-based STM systems do exist; these generally require that the data that should have transactional semantics be declared of some special type, such as "TransactionalInt". STM.NET is not like that—it provides transactional semantics for ordinary data transparently, simply by virtue of being accessed within the (dynamic) scope of a transaction.

This requires every read and write occurring in code that is executed within the transaction to be modified to make additional associated calls that acquire necessary locks, create and populate shadow copies, and so on. In our implementation, we modified the CLR's JIT compiler extensively to produce very different code to be executed within transactions. The atomic keyword (even if we've presented it via a delegate-based API) changes the language semantics at a pretty fundamental level.

Thus, I claim that we did change the language. In a .NET language like C#, the language semantics are implemented by a combination of the source-level language compiler, and its assumptions about the semantics of the MSIL that it emits—how the CLR runtime will execute that IL. We radically changed the CLR's interpretation of the bytecodes, so I would say that this changes the language.

In particular, say the CLR's JIT compiler encounters code like this:

```
try {
  <body>
}
catch (AtomicMarkerException) {}
```

The code within <body> (and, recursively, within methods it calls) is dynamically modified to ensure transactional semantics. I should emphasize that this has absolutely nothing to do with exception handling—it is purely a hack to identify an atomic block, since the try/catch construct is the only mechanism available in IL for identifying a lexically scoped block. In the long run, we would want something more like an explicit "atomic" block in the IL language. The delegate-based interface is implemented in terms of this ersatz atomic block.

In summary, the IL-level atomic block, however expressed, really does change the semantics of code that runs in it in a fundamental way. This is why STM.NET contains a new, significantly modified CLR runtime, not just changes to the BCL. If you took a stock CLR runtime and ran with the BCL from STM.NET, the result would not give you transactional semantics (in fact, I doubt it would work at all). *—Dr. Dave Detlefs, Architect, Common Language Runtime, Microsoft*

The syntax isn't quite as elegant as an atomic keyword would be, but C# has the power of anonymous methods to capture the block of code that would make up the body of the desired atomic block, and it can thus be executed under similar kinds of semantics. (Sorry, but as of this writing, the STM.NET incubation effort only supports C#. There is no technical reason why it couldn't be extended to all languages, but the STM.NET team only focused on C# for the first release.)

Getting Started with STM.NET

The first thing you'll need to do is download the Microsoft .NET Framework 4 beta 1 Enabled to use Software Transactional Memory V1.0 bits (a long-winded name, which I'll shorten to STM.NET BCL, or just STM.NET) from the DevLabs Web site. While you're there, download the STM.NET Documentation and Samples as well. The former is the actual BCL and STM.NET tools and supplemental assemblies, and the latter contains, among the documentation and sample projects, a Visual Studio 2008 template for building STM.Net applications.

Creating a new STM.NET-enabled application begins like any other app, in the New Project dialog (see **Figure 1**). Selecting the TMConsoleApplication template does a couple of things, some of which aren't entirely intuitive. For example, as of this writing, to execute against the STM.NET libraries, the .NET application's app.config requires this little bit of versioning legerdemain:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <requiredRuntime version="v4.0.20506"/>
  </startup>
  ...
</configuration>
```

Other settings will be present, but the requiredRuntime value is necessary to tell the CLR launcher shim to bind against the STM.NET version of the runtime. In addition, the TMConsoleApplication template binds the assembly against versions of the mscorlib and System.Transactions assemblies installed in the directory where STM.NET is installed, rather than the versions that come with the stock .NET Framework 3.0 or 3.5 CLR. This is necessary, when you think about it, because if STM.NET is going to provide transactional access for anything beyond just the code that you write, it's going to need to use its own copy of mscorlib. Plus, if it's going to interact correctly with other forms of transactions—such as the lightweight transactions provided by the Lightweight Transaction Manager (LTM)—it needs to have its own version of System.Transactions as well.

Other than that, an STM.NET application will be a traditional .NET application, written in C# and compiled to IL, linked against the rest of the unmodified .NET assemblies, and so on. STM.NET assemblies, like the COM+ and EnterpriseServices components of the last decade, will have a few more extensions in them describing transactional behaviors for the methods that interact with the STM.NET transactional behavior, but I'll get to that in time.

Hello, STM.NET

As with the Axum example in the September 2009 issue of *MSDN Magazine* (msdn.microsoft.com/magazine/ee412254), writing a traditional Hello World application as the starting point for STM.NET is actually harder than you might think at first, largely because if you write it without concern for transactions, it's exactly the same as the traditional C# Hello World. If you write it to take advantage of the STM.NET transactional behavior, you have to consider the fact that writing text to the console is, in fact, an un-undoable method (at least as far as STM.NET is concerned), which means that trying to roll back a Console.WriteLine statement is difficult.

So, instead, let's take a simple example from the STM.NET User Guide as a quick demonstration of the STM.NET bits. An object (called MyObject) has two private strings on it and a method to set those two strings to some pair of values:

```
class MyObject {
  private string m_string1 = "1";
  private string m_string2 = "2";

  public bool Validate() {
    return (m_string1.Equals(m_string2) == false);
  }
  public void SetStrings(string s1, string s2) {
    m_string1 = s1;
    Thread.Sleep(1);   // simulates some work
    m_string2 = s2;
  }
}
```

Because the assignment of the parameter to the field is itself an atomic operation, there's no concern around concurrency there. But just as with the BankAccount example shown earlier, you want either

Figure 2 **Manually Validating Atomic Updates to MyObject**

```
[AtomicNotSupported]
static void Main(string[] args) {
  MyObject obj = new MyObject();
  int completionCounter = 0; int iterations = 1000;
  bool violations = false;

  Thread t1 = new Thread(new ThreadStart(delegate {
    for (int i = 0; i < iterations; i++)
      obj.SetStrings("Hello", "World");
    completionCounter++;
  }));

  Thread t2 = new Thread(new ThreadStart(delegate {
    for (int i = 0; i < iterations; i++)
      obj.SetStrings("World", "Hello");
    completionCounter++;
  }));

  Thread t3 = new Thread(new ThreadStart(delegate {
    while (completionCounter < 2) {
      if (!obj.Validate()) {
        Console.WriteLine("Violation!");
        violations = true;
      }
    }
  }));

  t1.Start(); t2.Start(); t3.Start();
  while (completionCounter < 2)
    Thread.Sleep(1000);

  Console.WriteLine("Violations: " + violations);
...
```

Figure 3 **Validating MyObject with STM.NET**

```
class MyObject {
  private string m_string1 = "1";
  private string m_string2 = "2";

  public bool Validate() {
    bool result = false;
    Atomic.Do(() => {
      result = (m_string1.Equals(m_string2) == false);
    });
    return result;
  }

  public void SetStrings(string s1, string s2) {
    Atomic.Do(() => {
      m_string1 = s1;
      Thread.Sleep(1); // simulates some work
      m_string2 = s2;
    });
  }
}
```

both to be set or neither, and you don't want to see partial updates—one string being set, but not the other—during the set operation. You'll spawn two threads to blindly set the strings over and over again, and a third thread to validate the contents of the MyObject instance, reporting a violation in the event Validate returns false (see **Figure 2**).

The STM.NET environment needs some assistance in understanding whether methods called during an Atomic block are transaction-friendly.

Note that the way this example is constructed, validation fails if the two strings in obj are set to the same thing, indicating that Thread t1's SetStrings("Hello", "World") is partially updated (leaving the first "Hello" to match the second "Hello" set by t2).

A cursory glance at the SetStrings implementation shows that this code is hardly thread-safe. If a thread switch occurs in the middle (which is likely given the Thread.Sleep call, which will cause the currently-executing thread to give up its time slice), another thread could easily jump into the middle of SetStrings again, putting the MyObject instance into an invalid state. Run it, and with enough iterations, violations will start to appear. (On my laptop, I had to run it twice before I got the violations, proving that just because it runs without an error once doesn't mean the code doesn't have a concurrency bug.)

Modifying this to use STM.NET requires only a small change to the MyObject class, as shown in **Figure 3**.

As you can see, the only modification required was to wrap the bodies of Validate and SetStrings into atomic methods using the Atomic.Do operation. Now, when run, no violations appear.

Transactional Affinity

Observant readers will have noticed the [AtomicNotSupported] attribute at the top of the Main method in **Figure 2**, and perhaps wondered at its purpose, or even wondered if it served the same purpose as those attributes from the COM+ days. As it turns out, that's entirely correct: the STM.NET environment needs some assistance in understanding whether methods called during an Atomic block are transaction-friendly so that it can provide the necessary and desirable support for those methods.

Three such attributes are available in the current STM.NET release:

- AtomicSupported—the assembly, method, field or delegate supports transactional behavior and can be used inside or outside of atomic blocks successfully.
- AtomicNotSupported—the assembly, method, or delegate doesn't support transactional behavior and thus shouldn't be used inside of atomic blocks.
- AtomicRequired—the assembly, method, field or delegate not only supports transactional behavior, it should only be used inside of atomic blocks (thus guaranteeing that using this item will always be done under transactional semantics).

Technically there is a fourth, AtomicUnchecked, which signals to STM.NET that this item shouldn't be checked, period. It's intended as an escape hatch to avoid checking the code altogether.

The presence of the AtomicNotSupported attribute is what leads the STM.NET system to throw an AtomicContractViolationException when the following (naïve) code is attempted:

```
[AtomicNotSupported]
static void Main(string[] args) {
  Atomic.Do( () => {
    Console.WriteLine("Howdy, world!");
  });

  System.Console.WriteLine("Simulation done");
}
```

Because the System.Console.WriteLine method is not marked with AtomicSupported, the Atomic.Do method throws the exception when it sees the call in the atomic block. This bit of security ensures that only transaction-friendly methods are executed inside of the atomic block, and provides that additional bit of safety and security to the code.

Hello, STM.NET (Part Two)

What if you really, really want to write the traditional Hello World? What if you really want to print a line to the console (or write to a file, or perform some other non-transactional behavior) alongside two other transactional operations, but only print it out if both of those other operations succeed? STM.NET offers three ways to handle this situation.

Figure 4 **A Transactional Resource Manager**

```
public class TxAppender : TransactionalOperation {
  private TextWriter m_tw;
  private List<string> m_lines;

  public TxAppender(TextWriter tw) : base() {
    m_tw = tw;
    m_lines = new List<string>();
  }

  // This is the only supported public method
  [AtomicRequired]
  public void Append(string line) {
    OnOperation();

    try {
      m_lines.Add(line);
    }
    catch (Exception e) {
      FailOperation();
      throw e;
    }
  }

  protected override void OnCommit() {
    foreach (string line in m_lines) {
      m_tw.WriteLine(line);
    }
    m_lines = new List<string>();
  }

  protected override void OnAbort() {
    m_lines.Clear();
  }
}
```

First, you can perform the non-transactional operation outside the transaction (and only after the transaction commits) by putting the code inside of a block passed to Atomic.DoAfterCommit. Because the code inside that block will typically want to use data generated or modified from inside the transaction, DoAfterCommit takes a context parameter that is passed from inside the transaction to the code block as its only parameter.

Second, you can create a compensating action that will be executed in the event that the transaction ultimately fails, by calling Atomic.DoWithCompensation, which (again) takes a context parameter to marshal data from inside the transaction to the committing or compensating block of code (as appropriate).

Third, you can go all the way and create a Transactional Resource Manager (RM) that understands how to participate with the STM.NET transactional system. This is actually less difficult than it might seem—just inherit from the STM.NET class TransactionalOperation, which has OnCommit and OnAbort methods that you override to provide the appropriate behavior in either case. When using this new RM type, call OnOperation at the start of your work with it (effectively enlisting the resource into the STM.NET transaction). Then call FailOperation on it in the event that the surrounding operations fail.

Thus, if you want to transactionally write to some text-based stream, you can write a text-appending resource manager like the one shown in **Figure 4**. This then allows you—in fact, by virtue of the [Atomic-Required] attribute, requires you—to write to some text stream via the TxAppender while inside an atomic block (see **Figure 5**).

This is obviously the longer route and will be suitable only in certain scenarios. It could fail for some kinds of media types, but for the most part, if all the actual irreversible behavior is deferred to the OnCommit method, this will suffice for most of your in-process transactional needs.

Putting STM.NET to Work

Working with an STM system takes a little getting used to, but once you're acclimated, working without it can feel crippling. Consider some of the potential places where using STM.NET can simplify coding.

When working with other transacted resources, STM.NET plugs in to existing transacted systems quickly and easily, making Atomic.Do the sole source of transacted code in your system. The STM.NET examples demonstrate this in the TraditionalTransactions sample, posting messages to an MSMQ private queue and making it obvious that, when the Atomic block fails, no message is posted to the queue. This is probably the most obvious usage.

In dialog boxes—particularly for multi-step wizard processes or settings dialogs—the ability to roll back changes to the settings or dialog data members when the user hits the Cancel button is priceless.

Unit tests such as NUnit, MSTest, and other systems exert great effort to ensure that, when written correctly, tests cannot leak results from one test to the next. If STM.NET reaches production status, NUnit and MSTest can refactor their test case execution code to use STM transactions to isolate test results from each other, generating a rollback at the end of each test method, and thus eliminating any changes that might have been generated by the test. Even more, any test that calls out to an AtomicUnsupported method will be flagged at test execution time as an error, rather than silently leaking the test results to some medium outside the test environment (such as to disk or database).

Figure 5 Using TxAppender

```
public static void Test13() {
  TxAppender tracer =
    new TxAppender(Console.Out);
  Console.WriteLine(
    "Before transactions. m_balance= " +
    m_balance);

  Atomic.Do(delegate() {
    tracer.Append("Append 1:  " + m_balance);
    m_balance = m_balance + 1;
    tracer.Append("Append 2:  " + m_balance);
  });

  Console.WriteLine(
    "After transactions. m_balance= "
    + m_balance);

  Atomic.Do(delegate() {
    tracer.Append("Append 1:  " + m_balance);
    m_balance = m_balance + 1;
    tracer.Append("Append 2:  " + m_balance);
  });

  Console.WriteLine(
    "After transactions. m_balance= "
    + m_balance);
}
```

STM.NET can also be used in domain object property implementation. Although most domain objects have fairly simple properties, either assigning to a field or returning that field's value, more complex properties that have multiple-step algorithms run the risk of multiple threads seeing partial updates (if another thread calls the property during its set) or phantom updates (in the event another thread calls the property during its set, and the original update is eventually thrown away due to a validation error of some form).

Even more interesting, researchers outside of Microsoft are looking into extending transactions into hardware, such that someday, updating an object's field or a local variable could be a transaction guarded at the hardware level by the memory chip itself, making the transaction blindingly fast in comparison to today's methods.

However, as with Axum, Microsoft depends on your feedback to determine if this technology is worth pursuing and productizing, so if you find this idea exciting or interesting, or that it's missing something important to your coding practice, don't hesitate to let them know. ■

TED NEWARD *is a Principal with Neward and Associates, an independent firm specializing in .NET and Java enterprise systems. He has written numerous books, is a Microsoft MVP Architect, INETA speaker, and PluralSight instructor. Reach Ted at ted@tedneward.com, or read his blog at blogs.tedneward.com.*

THANKS to the following technical experts for reviewing this article: *Dave Detlefs and Dana Groff*

CHARLES PETZOLD

Line Charts with Data Templates

Despite the many advanced manifestations of computer graphics these days (including animation and 3-D), I suspect that the most important will forever be the basic visual representation of data in traditional charts built from bars, pies and lines.

A table of data may appear like a jumble of random numbers, but any trends or interesting information hidden within the figures become much more comprehensible when displayed in a chart.

With the Windows Presentation Foundation (WPF)—and its Web-based offshoot, Silverlight—we have discovered the advantages of defining graphical visuals in markup rather than code. Extensible Application Markup Language (XAML) is easier to alter than code, easier to experiment with and is more toolable than code, allowing us to define our visuals interactively and to play around with alternative approaches.

In fact, defining visuals entirely in XAML is so advantageous that WPF programmers will spend many hours writing code specifically to enable more powerful and flexible XAML. This is a phenomenon I call "coding for XAML," and it's one of several ways that WPF has altered our approach to application development.

Many of the most powerful techniques in WPF involve the ItemsControl, which is the basic control to display collections of items, generally of the same type. (One familiar control that derives from ItemsControl is the ListBox, which allows navigation and selection as well as display.)

An ItemsControl can be filled with objects of any type—even business objects that have no intrinsic textual or visual representation. The magic ingredient is a DataTemplate—almost always defined in XAML—that gives those business objects a visual representation based on the objects' properties.

In the March issue (msdn.microsoft.com/magazine/dd483292.aspx) I showed how to use ItemsControls and DataTemplates to define bar charts and pie charts in XAML. Originally I was going to include line charts in that article, but the importance (and difficulty) of line charts mandates that a whole column be devoted to the subject.

Code download available at code.msdn.microsoft.com/mag201001Charts.

With the Windows Presentation Foundation (and its Web-based offshoot, Silverlight), we have discovered the advantages of defining graphical visuals in markup rather than code.

Line Chart Issues

Line charts are actually forms of scatter plots—a Cartesian coordinate system with one variable on the horizontal axis and another on the vertical axis. One big difference with the line chart is that the values graphed horizontally are generally sorted. Very often these values are dates or times—that is, the line chart often shows the change in a variable over time.

The other big difference is that the individual data points are often connected with a line. Although this line is obviously a basic part of the line-chart visuals, it actually throws a big monkey wrench into the process of realizing the chart in XAML. The DataTemplate describes how each item in the ItemsControl is rendered; but connecting the items requires access to multiple points, ideally a PointCollection that can then be used with a Polyline element. The need to generate this PointCollection was the first hint that a custom class would be needed to perform pre-processing on the line-chart data.

More than other graphs, line graphs mandate that much more attention be paid to the axes. In fact, it makes sense for the horizontal and vertical axes themselves to be additional ItemsControls! Additional DataTemplates for these two other ItemsControls can then be used to define the formatting of the axis tick marks and labels entirely in XAML.

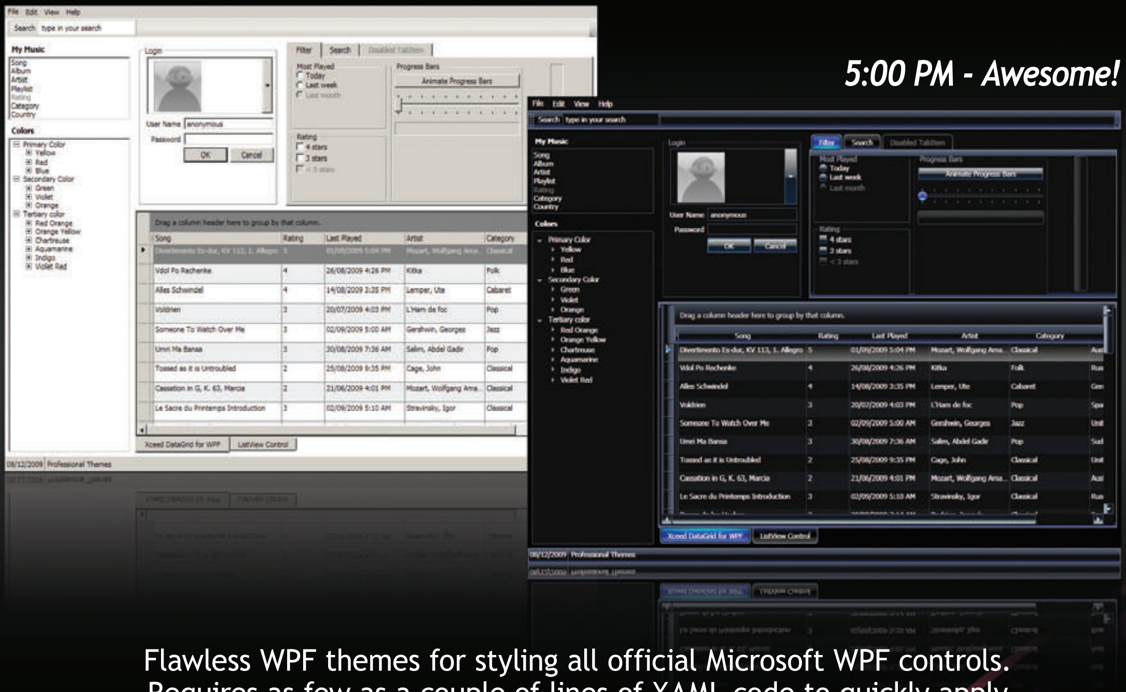
In summary, what you start out with is a collection of data items with two properties: one property corresponding to the horizontal axis and the other to the vertical axis. To realize a chart in XAML, you need to get certain items from this data. First, you need point objects for each data item (to render each data point). You also need a PointCollection of all data items (for the line connecting the points), and two additional collections containing sufficient information to render the horizontal and vertical axes in XAML, including data for the labels and offsets for positioning the labels and tick marks.

XCEED
Professional Themes
for WPF

Beautifully themed WPF apps in under 2 minutes.

4:58 PM - Ewww!

5:00 PM - Awesome!



Drag a column header here to group by that column.

Song	Rating	Last Played	Artist	Category
Divertimento Es-dur, KV 113, 1. Allegro	3	01/09/2009 5:04 PM	Mozart, Wolfgang Ama...	Classical
Vdol Po Rechenke	4	26/08/2009 4:26 PM	Kitka	Folk
Alles Schwindel	4	14/08/2009 3:35 PM	Lemper, Ute	Cabaret
Voldrien	3	20/07/2009 4:03 PM	L'Ham de foc	Pop
Someone To Watch Over Me	3	02/09/2009 5:00 AM	Gershwin, Georges	Jazz
Umn Ma Bansa	3	30/08/2009 7:36 AM	Salim, Abdel Gadir	Pop
Tossed as it is Untroubled	2	25/08/2009 9:35 PM	Cage, John	Classical
Cassation in G, K. 63, Marcia	2	21/06/2009 4:01 PM	Mozart, Wolfgang Ama...	Classical
Le Sacre du Printemps Introduction	3	02/09/2009 5:10 AM	Stravinsky, Igor	Classical

Flawless WPF themes for styling all official Microsoft WPF controls. Requires as few as a couple of lines of XAML code to quickly apply beautiful, uniform styles to your entire WPF application.

Try it live on xceed.com

XCEED
MULTI-TALENTED COMPONENTS

The calculation of these Point objects and offsets obviously requires some information: the width and height of the chart and the minimum and maximum values of the data graphed on the horizontal and vertical axes.

But that's not quite enough. Suppose the minimum value for the vertical axis is 127 and the maximum value is 232. In that case, you might want the vertical axis to actually extend from 100 to 250 with tick marks every 25 units. Or for this particular graph, you might want to always include 0, so the vertical axis extends from 0 to 250. Or perhaps you want the maximum value to always be a multiple of 100, so it goes from 0 to 300. If the values range from -125 to 237, perhaps you want 0 to be centered, so the axis might range from -300 to 300.

There are potentially many different strategies for determining what values the axes display, which then govern the calculation of the Point values associated with each data item. These strategies might be so varied that it makes sense to offer a "plug-in" option to define additional axis strategies as required for a particular chart.

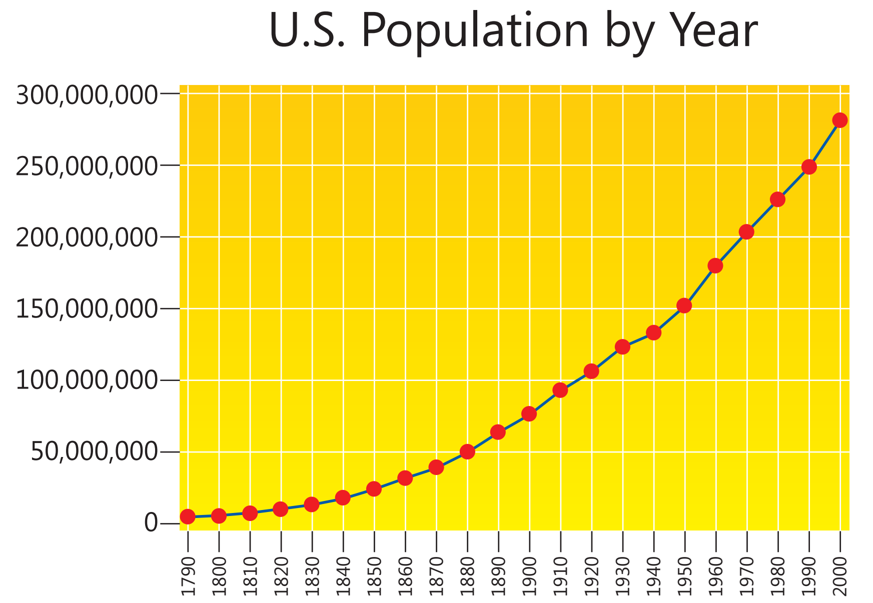


Figure 1 **The PopulationLineChart Display**

The First Attempt

Programming failures are sometimes just as instructive as programming successes. My first attempt to create a line-charting class accessible from XAML wasn't exactly a complete failure, but it was certainly headed in that direction.

I knew that to generate the collection of Point objects I would obviously need access to the collection of items in the ItemsControl, as well as the ActualWidth and ActualHeight of the control. For these reasons it seemed logical to derive a class from ItemsControl that I called LineChartItemsControl.

LineChartItemsControl defined several new read/write properties: HorizontalAxisPropertyName and VerticalAxisPropertyName provided the names of the items' properties that would be graphed. Four other new properties provided LineChartItemsControl with minimum and maximum values for the horizontal and vertical axes. (This was a very simple approach to handling the axes that I knew would have to be enhanced at a later time.)

The custom control also defined three read-only dependency properties for data binding in XAML: a property named Points of type PointCollection and two properties called HorizontalAxisInfo and VerticalAxisInfo for rendering the axes.

LineChartItemsControl overrode the OnItemsSourceChanged and OnItemsChanged methods to be informed whenever changes were occurring in the items collection, and it installed a handler for the SizeChanged event. It was then fairly straightforward to put together all the available information to calculate the three read-only dependency properties.

Actually using LineChartItemsControl in XAML, however, was a mess. The easy part was rendering the connected line. That was done with a Polyline element with its Points property bound to the Points property of LineChartItemsControl. But defining a DataTemplate that would position the individual data was very hard. The DataTemplate only has access to the properties of one particular data item. Through bindings, the DataTemplate can access the ItemsControl itself, but how do you get access to positioning information that corresponds to that particular data item?

My solution involved a RenderTransform set from a MultiBinding that contained both a RelativeSource binding and referenced a BindingConverter. It was so complex that the day after I had coded it, I couldn't quite figure out how it worked!

The complexity of this solution was a clear indication that I needed a whole different approach.

The Line Chart Generator in Practice

The reconceived solution was a class I called LineChartGenerator because it generates all the raw materials necessary to define the

Figure 2 **The Resources Section of PopulationLineChart**

```
<Window.Resources>
    <src:CensusData x:Key="censusData" />

    <charts:LineChartGenerator
            x:Key="generator"
            ItemsSource="{Binding Source={StaticResource censusData}}"
            Width="300"
            Height="200">

        <charts:LineChartGenerator.HorizontalAxis>
            <charts:AutoAxis PropertyName="Year" />
        </charts:LineChartGenerator.HorizontalAxis>

        <charts:LineChartGenerator.VerticalAxis>
            <charts:IncrementAxis PropertyName="Population"
                                  Increment="50000000"
                                  IsFlipped="True" />
        </charts:LineChartGenerator.VerticalAxis>
    </charts:LineChartGenerator>
</Window.Resources>
```

visuals of a chart entirely in XAML. One collection goes in (the actual business objects) and four collections come out—one for the data points, one for drawing the connected line and two more for the horizontal and vertical axes. This allows you to construct a chart in XAML that contains multiple ItemsControls (generally arranged in a four-by-four Grid, or larger if you want to include titles and other labels), each with its own DataTemplate to display these collections.

Let's see how this works in practice. (All downloadable source code is contained in a single Visual Studio project named LineChartsWith-DataTemplates. This solution has one DLL project named LineChar-tLib and three demonstration programs.)

The PopulationLineChart project contains a structure named CensusDatum that defines two properties of type int named Year and Population. The CensusData class derives from Observable-Collection of type CensusDatum and fills up the collection with U. S. decennial census data from the years 1790 (when the population was 3,929,214) through 2000 (281,421,906). **Figure 1** shows the resultant chart.

All the XAML for this chart is in the Window1.xaml file in the PopulationLineChart project. **Figure 2** shows the Resources section of this file. LineChartGenerator has its own ItemsSource property; in this example it's set to the CensusData object. It's also necessary to set the Width and Height properties here. (I realize this isn't an optimum place for these values, and not quite conducive to the preferred method of layout in WPF, but I couldn't work out a better solution.) These values indicate the interior dimensions of the chart excluding the horizontal and vertical axes.

Programming failures are sometimes just as instructive as programming successes.

LineChartGenerator also has two properties of type AxisStrategy named HorizontalAxis and VerticalAxis. AxisStrategy is an abstract class that defines several properties, including PropertyName where you indicate the property of the data objects you want graphed on this axis. In accordance with WPF's coordinate system, increasing values go from left to right and from top to bottom. Almost always you'll want to set the IsFlipped property on the vertical axis to True so increasing values go from bottom to top.

One of the classes that derives from AxisStrategy is IncrementAxis, which defines one property named Increment. With the Increment-Axis strategy, you specify what increment you want between the tick marks. The minimum and maximum are set as multiples of the increment. I've used IncrementAxis for the population scale.

Another class that derives from AxisStrategy is AutoAxis, which defines no additional properties of its own. I've used this one for the horizontal axis: All it does is use the actual values for the axis. (Another obvious AxisStrategy derivative that I haven't written is ExplicitAxis, where you supply a list of values to appear on the axis.)

Figure 3 **The Main ItemsControl for PopulationLineChart**

```
<ItemsControl ItemsSource="{Binding Source={StaticResource generator},
                                    Path=ItemPoints}">
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <Grid IsItemsHost="True" />
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>

    <ItemsControl.ItemTemplate>
        <DataTemplate>
            <Path Fill="Red" RenderTransform="2 0 0 2 0 0">
                <Path.Data>
                    <EllipseGeometry Center="{Binding Point}"
                                     RadiusX="4"
                                     RadiusY="4"
                                     Transform="0.5 0 0 0.5 0 0" />
                </Path.Data>
                <Path.ToolTip>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Text="{Binding Item.Year}" />
                        <TextBlock Text="{Binding Item.Population,
                            StringFormat=': {0:N0}'}" />
                    </StackPanel>
                </Path.ToolTip>
            </Path>
        </DataTemplate>
    </ItemsControl.ItemTemplate>
</ItemsControl>
```

The LineChartGenerator class defines two read-only dependency properties. The first is named Points of type PointCollection; use this property to draw the line that connects the points:

```
<Polyline Points="{Binding Source={StaticResource generator},
                           Path=Points}"
          Stroke="Blue" />
```

The second LineChartGenerator property is named ItemPoints of type ItemPointCollection. An ItemPoint has two properties, named Item and Point. Item is the original object in the collection—in this particular example, Item is an object of type CensusDatum. Point is the point where that item is to appear in the graph.

Figure 3 shows the ItemsControl that displays the main body of the chart. Notice that its ItemsSource is bound to the ItemPoints property of the LineChartGenerator. The ItemsPanel template is a Grid, and the ItemTemplate is a Path with an EllipseGeometry and a ToolTip. The Center property of the EllipseGeometry is bound to the Point property of the ItemPoint object, while the ToolTip accesses the Year and Population properties of the Item property.

You might be wondering about the Transform set on the Ellipse-Geometry object, which is offset by a RenderTransform property set on the Path element. This is a kludge: Without it, the ellipse at the far right was partially clipped, and I couldn't fix it with ClipToBounds.

The Polyline and this main ItemsControl share the same single-cell Grid whose Width and Height are bound to the values from LineChartGenerator:

```
<Grid Width="{Binding Source=
{StaticResource generator}, Path=Width}"
      Height="{Binding Source=
{StaticResource generator}, Path=Height}">
```

The Polyline is underneath the ItemsControl in this example.

The AxisStrategy class defines its own read-only dependency property named AxisItems, a collection of objects of type Axis-Item, which has two properties named Item and Offset. This is the collection used for the ItemsControl for each axis. Although the

Figure 4 **Markup for the Horizontal Axis of PopulationLineChart**

```
<ItemsControl Grid.Row="2"
              Grid.Column="1"
              Margin="4 0"
              ItemsSource="{Binding Source={StaticResource generator},
                                    Path=HorizontalAxis.AxisItems}">
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <Grid IsItemsHost="True" />
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>

    <ItemsControl.ItemTemplate>
        <DataTemplate>
            <StackPanel>
                <Line Y2="10" Stroke="Black" />
                <TextBlock Text="{Binding Item}"
                           FontSize="8"
                           LayoutTransform="0 -1 1 0 0 0"
                           RenderTransform="1 0 0 1 -6 1"/>

                <StackPanel.RenderTransform>
                    <TranslateTransform X="{Binding Offset}" />
                </StackPanel.RenderTransform>
            </StackPanel>
        </DataTemplate>
    </ItemsControl.ItemTemplate>
</ItemsControl>
```

Item property is defined to be of type object, it will actually be the same type as the property associated with that axis. Offset is a distance from the top or left.

Figure 4 shows the ItemsControl for the horizontal axis; the vertical axis is similar. The ItemsSource property of the ItemsControl is bound to the AxisItems property of the HorizontalAxis property of the LineChartGenerator. The ItemsControl is thus filled with objects of type AxisItem. The Text property of the TextBlock is bound to the Items property, and the Offset property is used to translate the tick mark and text along the axis.

Because these three ItemsControls are simply sitting in three cells of a Grid, it's the responsibility of the person designing the layout in XAML to make sure they align correctly. Any borders or margins or padding applied to these controls must be consistent. The ItemsControl in **Figure 4** has a horizontal margin of 4; the ItemsControl for the vertical axis has a vertical margin of 4. I chose these values to correspond to the BorderThickness and Padding of a Border surrounding the single-cell Grid that contains the Polyline and the chart itself:

```
<Border Grid.Row="1"
        Grid.Column="1"
        Background="Yellow"
        BorderBrush="Black"
        BorderThickness="1"
        Padding="3">
```

Data Type Consistency

The LineChartGenerator class itself is not very interesting. It assumes that the ItemsSource collection is already sorted, and is mostly devoted to making sure that everything gets updated when the ItemsSource property changes. If the collection set to ItemsSource implements ICollectionChanged, the chart is also updated when items are added to or removed from the collection. If the items in the collection implement INotifyPropertyChanged, the chart is updated when the items themselves change.

Most of the real work actually goes on in AxisStrategy and derived classes. These AxisStrategy derivatives are the classes you set to LineChartGenerator's HorizontalAxis and VerticalAxis properties.

AxisStrategy itself defines the important PropertyName property that indicates which property of the objects being charted is associated with that axis. AxisStrategy uses reflection to access that particular property of the objects in the collection. But just accessing that property isn't enough. AxisStrategy (and its derivatives) need to perform calculations on the values of this property to obtain Point objects and tick offsets. These calculations include multiplication and division.

The necessity of calculations strongly implies that the properties being graphed must be numeric types—integers or floating point. Yet one extremely common data type commonly used on the horizontal axis of line charts is not a number at all, but a date or a time. In the Microsoft .NET Framework, we're talking about an object of type DateTime.

What do all the numeric data types and DateTime have in common? They all implement the IConvertible interface, which means that they all contain a bunch of methods that convert them into one another, and they are all usable with the same-named methods in the static Convert class. Therefore, it seemed reasonable to me to require that the properties being charted implement IConvertible. AxisStrategy (and its derivatives) could then simply convert the property values to doubles to perform the necessary calculations.

However, I soon discovered that properties of type DateTime actually *cannot* be converted to doubles using either the ToDouble method or the Convert.ToDouble static method. This

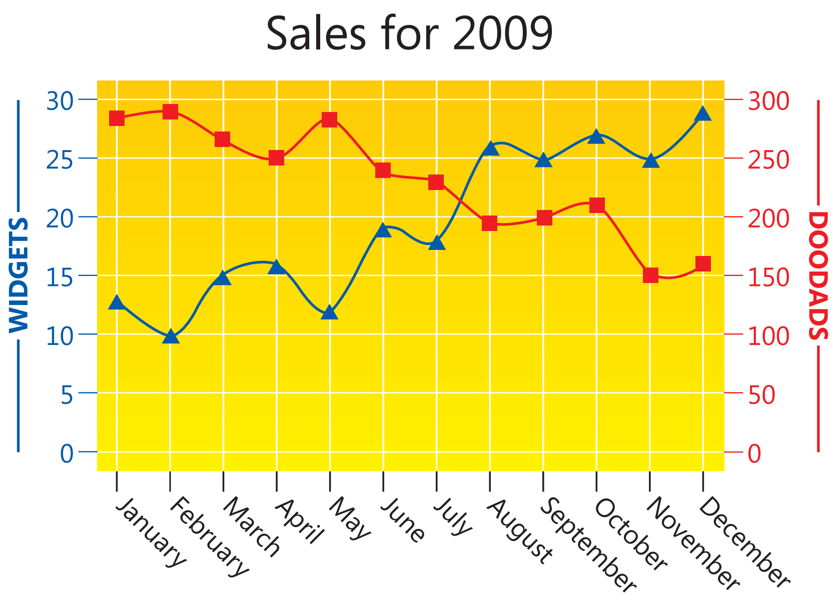


Figure 5 **The SalesByMonth Display**

meant that properties of type DateTime really had to be handled with special logic, which fortunately didn't turn out to be a big deal. The Ticks property defined by DateTime is a 64-bit integer, which can be converted to a double; a double can be converted back to a DateTime by first converting it to a 64-bit integer and then passing that value to a DateTime constructor. A little experimentation revealed that the round-trip conversion was accurate to the millisecond.

I soon discovered that properties of type DateTime actually *cannot* be converted to doubles using either the ToDouble method or the Convert.ToDouble static method.

AxisStrategy has a Recalculate method that loops through all the items in its parent's ItemsSource collection, converts the specified property of each object to a double and determines the minimum and maximum values. AxisStrategy defines three properties that potentially affect these two values: Margin (which allows the minimum and maximum to be a little beyond the range of actual values), IncludeZero (so that the axis always includes the value zero even if all the values are greater than zero or less than zero), and IsSymmetricAroundZero, which means that the axis maximum should be positive, and the minimum should be negative, but they should have the same absolute values.

After those adjustments, AxisStrategy calls the abstract CalculateAxisItems method:

```
protected abstract void CalculateAxisItems(Type propertyType, ref double
minValue, ref double maxValue);
```

The first argument is the type of the properties corresponding to that axis. Any class that derives from AxisStrategy must implement this method and use the opportunity to define the items and offsets that constitute its AxisItems collection.

It's very likely that CalculateAxisItems will also set new minimum and maximum values. When CalculateAxisItems returns, AxisStrategy then uses these values together with the width and height of the chart to calculate Point values for all the items.

XML Data Sources

Besides dealing with numeric properties and properties of type DateTime, AxisStategy also must handle the case when the items in the ItemsSource collection are of type XmlNode. This is what the collection contains when ItemsSource is bound to an XmlDataProvider, either referencing an external XML file or an XML data island within the XAML file.

AxisStrategy uses the same conventions as DataTemplates: A name by itself refers to an XML element, and a name preceded by an @ sign is an XML attribute. AxisStrategy obtains these values as strings. Just in case they are actually dates or times, AxisStrategy first attempts to convert these strings into DateTime objects before converting them into doubles. DateTime.TryParseExact is used for this job, and only for the invariant-culture formatting specifications of "R", "s", "u" and "o".

The SalesByMonth project demonstrates graphing XML data and a few other features. The Window1.xaml file contains an XmlDataProvider with data for 12 months of fictitious sales of two products named Widgets and Doodads:

```
<XmlDataProvider x:Key="sales"
                 XPath="YearSales">
    <x:XData>
        <YearSales xmlns="">
            <MonthSales Date="2009-01-01T00:00:00">
                <Widgets>13</Widgets>
                <Doodads>285</Doodads>
            </MonthSales>

                        ...

            <MonthSales Date="2009-12-01T00:00:00">
                <Widgets>29</Widgets>
                <Doodads>160</Doodads>
            </MonthSales>
        </YearSales>
    </x:XData>
</XmlDataProvider>
```

The Resources section also contains two very similar LineChartGenerator objects for the two products. Here's the one for the Widgets:

```
<charts:LineChartGenerator
                x:Key="widgetsGenerator"
                ItemsSource=
                "{Binding Source={StaticResource sales},
                                      XPath=MonthSales}"
                Width="250" Height="150">
    <charts:LineChartGenerator.HorizontalAxis>
        <charts:AutoAxis PropertyName="@Date" />
    </charts:LineChartGenerator.HorizontalAxis>

    <charts:LineChartGenerator.VerticalAxis>
        <charts:AdaptableIncrementAxis
        PropertyName="Widgets"
        IncludeZero="True"
        IsFlipped="True" />
    </charts:LineChartGenerator.VerticalAxis>
</charts:LineChartGenerator>
```

Notice that the horizontal axis is associated with the XML attribute of Date. The vertical axis is of type AdaptableIncrementAxis, which derives from AxisStrategy and defines two additional properties:

- Increments of type DoubleCollection
- MaximumItems of type int

The Increments collection has default values 1, 2 and 5, and the MaximumItems property has a default value of 10. The SalesByMonth project simply uses those defaults. AdaptableIncrementAxis determines the optimum increment between tick marks so the number of axis items does not exceed MaximumItems. With the default settings, it tests increment values of 1, 2 and 5, and then 0, 20 and 50, and then 100, 200 and 500 and so forth. It will also go in the opposite direction: testing increments of 0.5, 0.2, 0.1 and so forth.

You can fill the Increments property of AdaptableIncrementAxis with other values, of course. If you want the increment to always be a multiple of 10, just use the single value 1. An alternative to 1, 2 and 5 that might be more appropriate for some situations is 1, 2.5 and 5.

The canonical spline—also known as the cardinal spline—is part of Windows Forms but did not make it into the WPF.

AdaptableIncrementAxis (or something like it of your own invention) is probably the best choice when the numeric values of an axis are unpredictable, particularly when the chart contains data that is dynamically changing or growing in overall size. Because the Increments property of AdaptableIncrementAxis is of type DoubleCollection, it's unsuitable for DateTime values. I describe an alternative for DateTime later in this column.

The XAML file in the SalesByMonth project defines two LineChartGenerator objects for the two products, which then allows a composite chart as shown in **Figure 5**.

This option of creating a composite chart did not require anything special in the classes that make up LineChartLib. All the code does is generate collections that can then be handled flexibly in XAML.

To accommodate all the labels and axes, the entire chart is realized in a Grid of four rows and five columns containing five ItemsControls—two for the two collections of data items in the chart itself, two for the axis scales on the left and right, and one more for the horizontal axis.

The color-coding to distinguish the two products is simple to implement in XAML. But notice also that the two products are further distinguished by triangular and square data points. The triangular items are rendered by this DataTemplate:

```
<DataTemplate>
    <Path Fill="Blue"
          Data="M 0 -4 L 4 4 -4 4Z">
        <Path.RenderTransform>
            <TranslateTransform X="{Binding Point.X}"
                                Y="{Binding Point.Y}" />
        </Path.RenderTransform>
    </Path>
</DataTemplate>
```

In a real-life example you might use shapes actually associated with the two products, or even little bitmaps.

The line that connects the points in this example is not a standard Polyline element but instead a custom Shape derivative named CanonicalSpline. (The canonical spline—also known as the cardinal spline—is part of Windows Forms but did not make it into the WPF. Every pair of points is connected by a curve that depends algorithmically on the two additional points surrounding the pair of points.) It's also possible to write other custom classes for this purpose, perhaps one that performs least-squares interpolation on the points and displays the result.

The HorizontalAxis.AxisItems property of the LineChartChartGenerator is an ObservableCollection of type DateTime, which means that the items can be formatted using the StringFormat feature of the Binding class and standard date/time formatting strings.

Figure 6 **The DataTemplate for the Horizontal Axis of TemperatureHistory**

```
<DataTemplate>
    <StackPanel HorizontalAlignment="Left">
        <Line Y2="10" Stroke="Black" />

        <TextBlock Name="txtblk"
                   RenderTransform="1 0 0 1 -4 -4">
            <TextBlock.LayoutTransform>
                <RotateTransform Angle="45" />
            </TextBlock.LayoutTransform>
        </TextBlock>

        <StackPanel.RenderTransform>
            <TranslateTransform X="{Binding Offset}" />
        </StackPanel.RenderTransform>
    </StackPanel>

    <DataTemplate.Triggers>
        <DataTrigger Binding="{Binding Source={StaticResource generator},
                                       Path=HorizontalAxis.
                                       DateTimeInterval}"
                     Value="Second">
            <Setter TargetName="txtblk" Property="Text"
                 Value="{Binding Item, StringFormat=h:mm:ss d MMM yy}" />
        </DataTrigger>
        <DataTrigger Binding="{Binding Source={StaticResource generator},
                                       Path=HorizontalAxis.
                                       DateTimeInterval}"
                     Value="Minute">
            <Setter TargetName="txtblk" Property="Text"
                    Value="{Binding Item, StringFormat=h:mm d MMM yy}" />
        </DataTrigger>
        <DataTrigger Binding="{Binding Source={StaticResource generator},
                                       Path=HorizontalAxis.
                                       DateTimeInterval}"
                     Value="Hour">
            <Setter TargetName="txtblk" Property="Text"
                 Value="{Binding Item, StringFormat='h tt, d MMM yy'}" />
        </DataTrigger>
        <DataTrigger Binding="{Binding Source={StaticResource generator},
                                       Path=HorizontalAxis.
                                       DateTimeInterval}"
                     Value="Day">
            <Setter TargetName="txtblk" Property="Text"
                    Value="{Binding Item, StringFormat=d}" />
        </DataTrigger>
        <DataTrigger Binding="{Binding Source={StaticResource generator},
                                       Path=HorizontalAxis.
                                       DateTimeInterval}"
                     Value="Month">
            <Setter TargetName="txtblk" Property="Text"
                    Value="{Binding Item, StringFormat=MMM yy}" />
        </DataTrigger>
        <DataTrigger Binding="{Binding Source={StaticResource generator},
                                       Path=HorizontalAxis.
                                       DateTimeInterval}"
                     Value="Year">
            <Setter TargetName="txtblk" Property="Text"
                    Value="{Binding Item, StringFormat=MMMM}" />
        </DataTrigger>
    </DataTemplate.Triggers>
</DataTemplate>
```

The DataTemplate for the horizontal axis uses the "MMMM" formatting string to display whole month names:

```
<DataTemplate>
    <StackPanel HorizontalAlignment="Left">
        <Line Y2="10" Stroke="Black" />
        <TextBlock Text="{Binding Item, StringFormat=MMMM}"
                   RenderTransform="1 0 0 1 -4 -4">
            <TextBlock.LayoutTransform>
                <RotateTransform Angle="45" />
            </TextBlock.LayoutTransform>
        </TextBlock>

        <StackPanel.RenderTransform>
            <TranslateTransform X="{Binding Offset}" />
        </StackPanel.RenderTransform>
    </StackPanel>
</DataTemplate>
```

Dates and Times

The use of DateTime objects on the horizontal axis of a line chart is so common that it's worth spending effort coding an AxisStrategy to deal specifically with these objects. Some line charts accumulate data such as stock prices or environmental readings, perhaps adding a new item every hour or so, and it would be nice to have an AxisStrategy that adapts itself depending on the range of DateTime values among the graphed items.

My stab at such a class is called AdaptableDateTimeAxis, and it's intended to accommodate DateTime data over a wide range from seconds to years.

AdaptableDateTimeAxis has a MaximumItems property (with a default setting of 10) and six collections called SecondIncrements, MinuteIncrements, HourIncrements, DayIncrements, MonthIncrements and YearIncrements. The class systematically tries to find an increment between tick points so that the number of items does not exceed MaximumItems. With the default settings, AdaptableDateTimeAxis will test increments of 1 second, 2 seconds, 5, 15 and 30 seconds, then 1, 2, 5, 15 and 30 minutes, then 1, 2, 4, 6 and 12 hours, 1, 2, 5 and 10 days, and 1, 2, 4 and 6 months. Once it gets up to years, it tries 1, 2 and 5 years, then 10, 20 and 50, and so forth.

AdaptableDateTimeAxis also defines a read-only dependency property named DateTimeInterval—also the name of an enumeration with members Second, Minute, Hour and so forth—that indicates the units of the axis increments determined by the class. This property allows DataTriggers to be defined in XAML that alter the DateTime formatting based on the increment. **Figure 6** shows a sample DataTemplate that performs such formatting selection.

That template is from the TemperatureHistory project, which accesses the Web site of the National Weather Service to obtain hourly temperature readings in Central Park in New York City. **Figure 7** shows the TemperatureHistory display after the program ran for several hours; **Figure 8** shows it after several days.

Of course, my line-charting classes aren't entirely flexible—for example, currently there is no way to independently draw tick marks not associated with text labels—but I think they illustrate a viable and powerful approach to providing sufficient information to define line-chart visuals entirely in XAML. ■

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). His Web site is charlespetzold.com.*

THANKS to the following technical expert for reviewing this article: *David Teitelbaum*

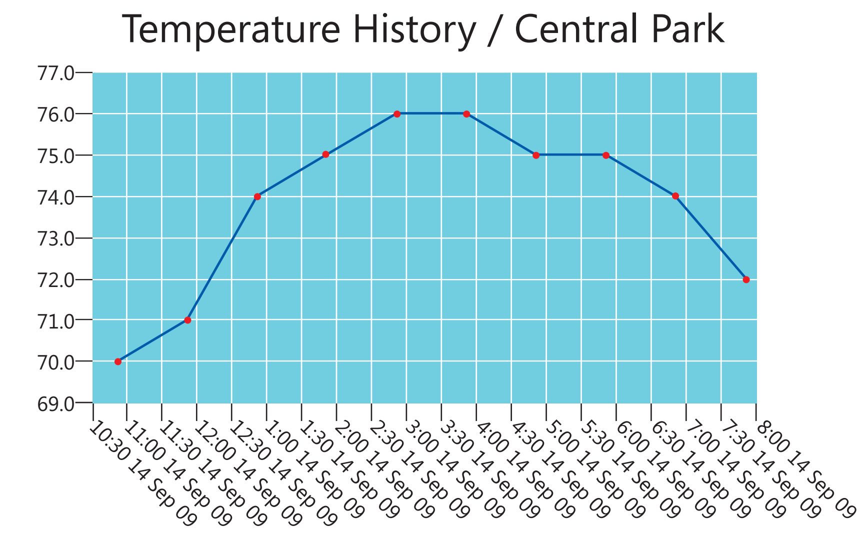


Figure 7 **The TemperatureHistory Display with Hours**

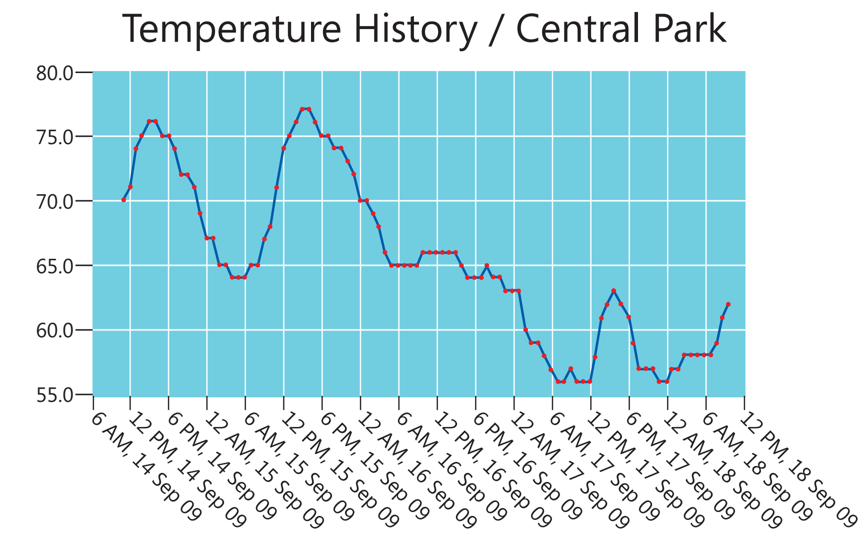


Figure 8 **The TemperatureHistory Display with Days**

JUVAL LOWY

Discover a New WCF with Discovery

All the Windows Communication Foundation (WCF) calls possible with the Microsoft .NET Framework 3.5 share two constraints. First, the port or pipe assigned to the service must be available. The application developer or administrator literally has to guess or have some way of reserving them. Second, the client must *a priori* know the address of the service endpoints, both the port number and the service machine, or the pipe name.

It would be great if the service could use any available address. The client, in turn, would need to discover that address at runtime. In fact, there is an industry standard-based solution that stipulates how that discovery stakes place. That solution, called simply *discovery* (and its supporting mechanisms), is the subject of this column. I will also introduce several useful tools and helper classes. The source code for these is available at code.msdn.

Address Discovery

Discovery relies on the User Datagram Protocol (UDP). Unlike the Transmission Control Protocol (TCP), UDP is a connectionless protocol, and no direct connection is required between the packets sender and the receiver. The client uses UDP to broadcast discovery requests for any endpoint supporting a specified contract type. These requests are received by dedicated discovery endpoints that the services support. The implementation of the discovery endpoint responds back to the client with the address of the service endpoints that support the specified contract. Once the client discovers the services, it continues to invoke them as with regular WCF calls. This sequence is illustrated in **Figure 1**.

Much like the Metadata Exchange (MEX) endpoint, WCF offers a standard discovery endpoint with the type UdpDiscoveryEndpoint:

```
public class DiscoveryEndpoint : ServiceEndpoint
{...}
public class UdpDiscoveryEndpoint : DiscoveryEndpoint
{...}
```

The service can have the host implement that endpoint by adding the ServiceDiscoveryBehavior to the collections of behaviors supported by the service. You can do that programmatically like this:

```
ServiceHost host = new ServiceHost(...);
host.AddServiceEndpoint(new UdpDiscoveryEndpoint());
ServiceDiscoveryBehavior discovery = new ServiceDiscoveryBehavior();
host.Description.Behaviors.Add(discovery);
host.Open();
```

Figure 2 shows how to add the discovery endpoint and the discovery behavior using the service config file.

Dynamic Addresses

Discovery is independent of how exactly the service host defines its endpoints. However, what if the client is expected to use discovery

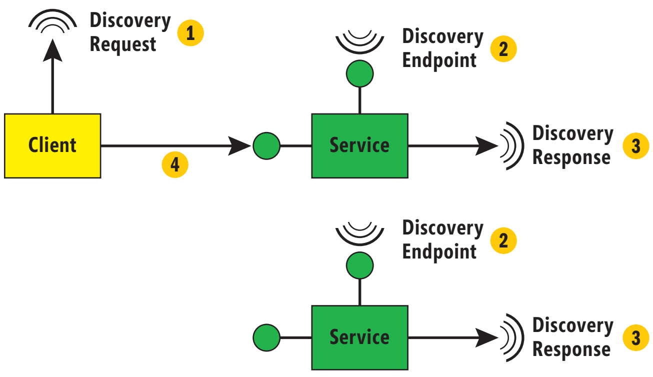


Figure 1 **Address Discovery over UDP**

to find the service address? In that case, the service is at liberty to configure its endpoint addresses on the fly, dynamically, based on any available port or pipe.

To automate using dynamic addresses, I wrote the DiscoveryHelper static helper class with the two properties AvailableIpcBaseAddress and AvailableTcpBaseAddress:

```
public static class DiscoveryHelper
{
   public static Uri AvailableIpcBaseAddress
   {get;}
   public static Uri AvailableTcpBaseAddress
   {get;}
}
```

Implementing AvailableIpcBaseAddress is straightforward—because any uniquely named pipe will do, the property uses a new globally unique identifier (GUID) to name the pipe. Implementing AvailableTcpBaseAddress is done by finding an available TCP port via opening port zero.

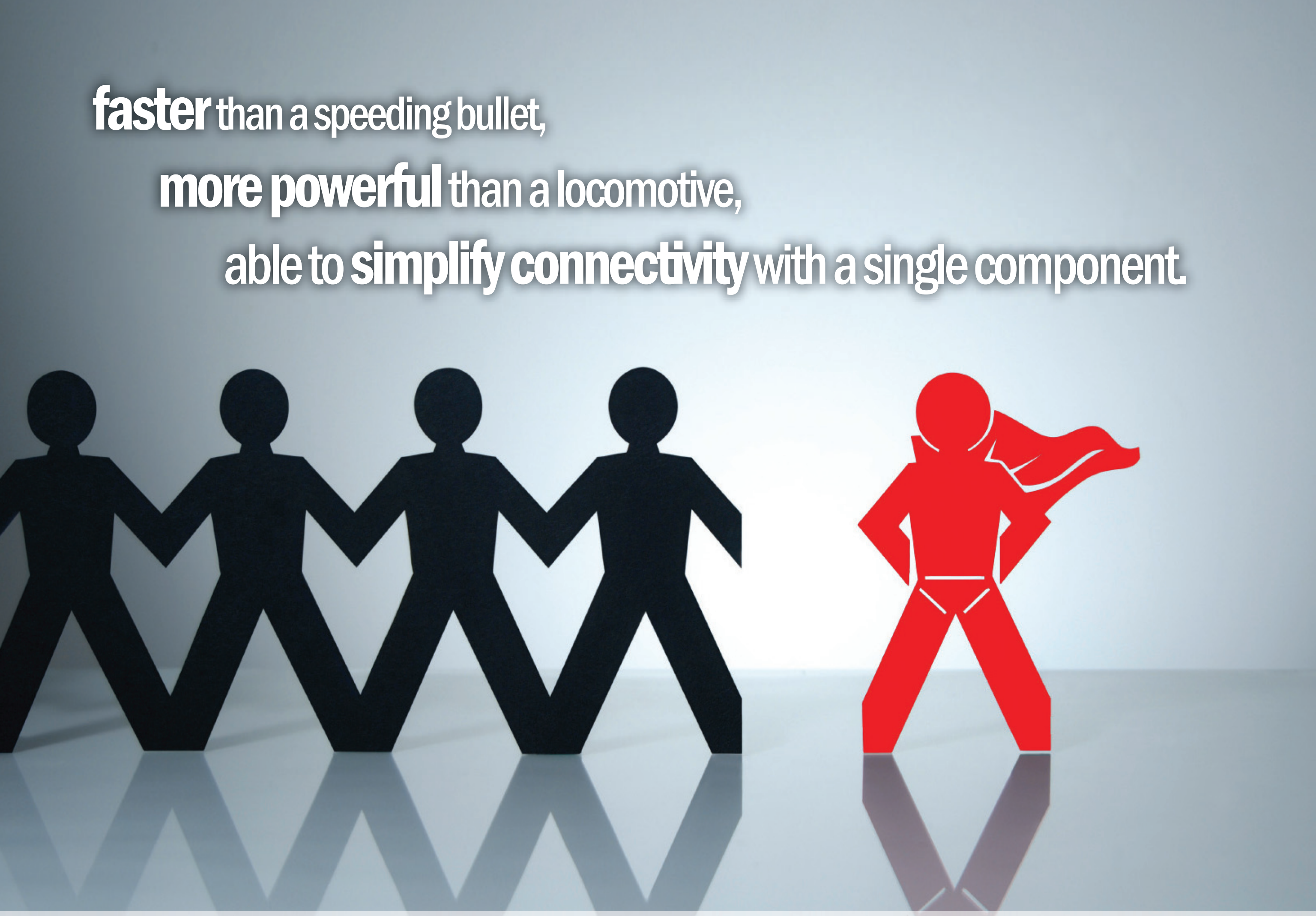
Figure 3 shows how to use AvailableTcpBaseAddress.

If all you want is the dynamic base address for your service, the code in **Figure 3** is less than perfect, because it still requires you to add discovery, either in the config file or programmatically. You can streamline these steps with my EnableDiscovery host extension, defined as:

```
public static class DiscoveryHelper
{
   public static void EnableDiscovery(this ServiceHost host,bool
enableMEX = true);
}
```

Send your questions and comments for Lowy to mmnet30@microsoft.com.

Code download available at code.msdn.microsoft.com/mag2010WCF.


faster than a speeding bullet,
more powerful than a locomotive,
able to simplify connectivity with a single component.

The Professional Developer's #1 Choice for Communications, Security, & E-Business Components

Internet Communications

IP*Works! Products

TCP/IP	**Secure SNMP**
SSL	**Zip**
S/MIME	**EDI AS2**
Secure Shell	

Our flagship product line. The result of more than a decade of research and development in building Internet connectivity tools for professional software developers.

Components for: Web and Web Services, Email and News, File Transfer, Sockets and Streaming, Remote Access, Instant Messaging, SSL and Secure Shell Security, Certificate Management & Creation, S/Mime Encryption, SNMP Network Management, File & Streaming Compression, and E-Business (B2B). Transactions.

Internet Business

IBiz Integrator Products

QuickBooks	**First Data**
E-Payment	**USPS**
E-Banking	**FedEx**
Vital/TSYS	**Amazon**
Paymentech	**PayPal**

The IBiz product line provides small and medium-sized companies with enterprise-class software components for Internet business integration. Built on top of our award winning IP*Works! tools, these easy-to-use components enable developers to rapidly integrate common business processes.

Components for: Accounting Integration (QuickBooks), Credit Card Processing, ACH / E-Check Processing, Shipping and Tracking, Banking &Financial Transactions, and Services Integration.

Enterprise Adapters

BizTalk & SQL Server Adapters

AS2 / EDI-INT	**GISB / NAESB**
SFTP / FTPS	**OFTP**
XMPP (Jabber)	**Secure Shell**
SMS Paging	**Secure Email**
AWS Integration	

The /n software Adapters extend Microsoft BizTalk and SQL Server with advanced Internet communications, security, and e-business capabilities, including robust, highly scalable, fully integrated implementations of B2B messaging and secure communications protocols.

Components for: EDI-INT AS2 Integration, Secure File Transfer, Secure Shell Remote Execution, Secure Email, RosettaNet Connectivity, OFTP Integration, NAESB / GISB Messaging, and more.

WWW.NSOFTWARE.COM - [ASK ABOUT OUR MSDN-STYLE ALL INCLUSIVE RED CARPET SUBSCRIPTIONS]

/n software

When using EnableDiscovery there is no need for programmatic steps or a config file:

```
Uri baseAddress = DiscoveryHelper.AvailableTcpBaseAddress;
ServiceHost host = new ServiceHost(typeof(MyService),baseAddress);
host.EnableDiscovery();
host.Open();
```

If the host has not already defined endpoints for the service, EnableDiscovery will add the default endpoints. EnableDiscovery will also default to adding to the service the MEX endpoint on its base addresses.

Client-Side Steps

The client uses the DiscoveryClient class to discover all endpoint addresses of all services that support a specified contract:

```
public sealed class DiscoveryClient : ICommunicationObject
{
    public DiscoveryClient();
    public DiscoveryClient(string endpointName);
    public DiscoveryClient(DiscoveryEndpoint discoveryEndpoint);
    public FindResponse Find(FindCriteria criteria);
   //More members
}
```

Logically, DiscoveryClient is a proxy to the discovery endpoint. Like all proxies, the client must provide the proxy's constructor with the information about the target endpoint. The client can use a config file to specify the endpoint or programmatically provide the standard UDP discovery endpoint for that purpose, since no further details (such as address or binding) are required. The client then calls the Find method, providing it with the contract type to discover via an instance of FindCriteria:

```
public class FindCriteria
{
   public FindCriteria(Type contractType);
   //More members
}
```

Find returns an instance of FindResponse, which contains a collection of all the discovered endpoints:

```
public class FindResponse
{
   public Collection<EndpointDiscoveryMetadata> Endpoints
   {get;}
   //More members
}
```

Each endpoint is represented by the EndpointDiscovery-Metadata class:

```
public class EndpointDiscoveryMetadata
{
   public EndpointAddress Address
  {get;set;}
   //More members
}
```

Figure 2 **Adding Discovery Endpoint in Config**

```
<services>
   <service name = "MyService">
      <endpoint
         kind = "udpDiscoveryEndpoint"
      />
      ...
   </service>
</services>
<behaviors>
   <serviceBehaviors>
      <behavior>
         <serviceDiscovery/>
      </behavior>
   </serviceBehaviors>
</behaviors>
```

Figure 3 **Using Dynamic Addresses**

```
Uri baseAddress = DiscoveryHelper.AvailableTcpBaseAddress;
ServiceHost host = new ServiceHost(typeof(MyService),baseAddress);
host.AddDefaultEndpoints();
host.Open();
<service name = "MyService">
   <endpoint
      kind = "udpDiscoveryEndpoint"
   />
</service>
<serviceBehaviors>
   <behavior>
      <serviceDiscovery/>
   </behavior>
</serviceBehaviors>
```

Figure 4 **Discovering and Invoking an Endpoint**

```
DiscoveryClient discoveryClient =
   new DiscoveryClient(new UdpDiscoveryEndpoint());
FindCriteria criteria = new FindCriteria(typeof(IMyContract));
FindResponse discovered = discoveryClient.Find(criteria);
discoveryClient.Close();
//Just grab the first found
EndpointAddress address = discovered.Endpoints[0].Address;
Binding binding = new NetTcpBinding();
IMyContract proxy =
   ChannelFactory<IMyContract>.CreateChannel(binding,address);
proxy.MyMethod();
(proxy as ICommunicationObject).Close();
```

The main property of the EndpointDiscoveryMetadata is Address, which finally contains the discovered endpoint address. **Figure 4** shows how a client can use these types in conjunction to discover the endpoint address and invoke the service.

There are several noteworthy problems with **Figure 4**.

While the client may discover multiple endpoints supporting the desired contract, it has no logic to resolve which one to invoke. It simply invokes the first one in the returned collection.

Discovery is geared toward addresses only. There's no information about which binding to use to invoke the service. **Figure 4** simply hardcodes the use of the TCP binding. The client will have to repeat these minute steps over and over every time it needs to discover the service address.

Discovery takes time. By default, Find will wait for 20 seconds for the services to respond to the UDP discovery request. Such a delay makes discovery inadequate for use in many applications, certainly when the application performs a high volume of tight calls. While you could shorten that timeout, if you do so, you run the risk of not discovering any or all of the services. DiscoveryClient does offer an asynchronous discovery, but that is of no use for a client that needs to invoke the service before continuing with its execution.

You will see several approaches to addressing these problems in this column.

Scopes

The use of discovery implies a somewhat loose relationship between the client and the service or services it discovers. This presents another set of problems—how can the client know it has discovered the right endpoint? When multiple compatible endpoints are discovered, which one should the client invoke?

Clearly, there is a need for some mechanism that will help the client filter the results of discovery. This is exactly what scopes are about. A scope is merely a valid URL associated with the endpoint. The service can associate a scope or even multiple scopes with each of its endpoints. The scopes are bundled along with the addresses in the response to the discovery request. In turn, the client can filter the discovered addresses based on the scopes found or, better yet, try to find only relevant scopes in the first place.

Scopes are immensely useful in customizing discovery and in adding sophisticated behavior to your application, especially when writing a framework or administration tools. The classic use for scopes is to enable the client to distinguish among polymorphic services from different applications. However, this is somewhat of a rare occurrence. I find scopes handy when it comes to distinguishing among endpoint types in the same application.

For example, suppose for a given contract you have multiple implementations. You have the operational mode used in production and the simulation mode used in testing or diagnostics. Using scopes, the client can pick and choose the correct implementation type needed, and different clients never conflict with one another by consuming one another's services. You can also have the same client pick up a different endpoint based on the context of the invocation. You could have endpoints for profiling, debugging, diagnostics, testing, instrumentation and so on.

The host assigns scopes on a per-endpoint basis using the EndpointDiscoveryBehavior class. For example, to apply across all endpoints, use a default endpoint behavior:

```
<endpointBehaviors>
   <behavior>
      <endpointDiscovery>
         <scopes>
            <add scope = "net.tcp://MyApplication"/>
         </scopes>
      </endpointDiscovery>
   </behavior>
</endpointBehaviors>
```

You apply scopes discretely, based on the type of service, by assigning the behaviors explicitly per endpoint, as shown in **Figure 5**. A single discovery behavior can list multiple scopes:

```
<endpointDiscovery>
   <scopes>
      <add scope = "net.tcp://MyScope1"/>
      <add scope = "net.tcp://MyScope2"/>
   </scopes>
</endpointDiscovery>
```

If an endpoint has multiple scopes associated with it, when the client tries to discover the endpoint based on scope matching, the

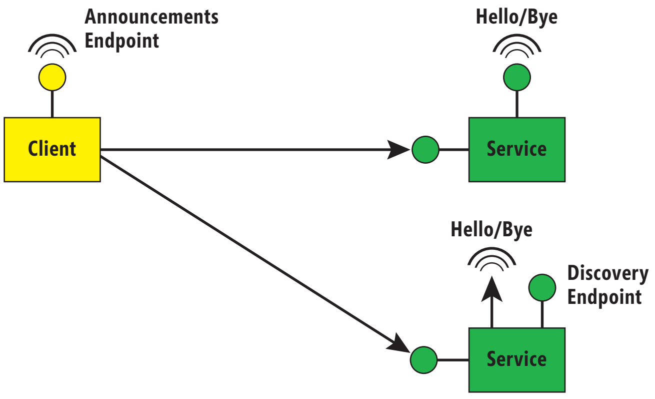


Figure 6 **The Announcement Architecture**

Figure 5 **Explicit Behavior Assignment**

```
<service name = "MySimulator">
   <endpoint behaviorConfiguration = "SimulationScope"
      ...
   />
   ...
</service>
...
   <behavior name = "SimulationScope">
      <endpointDiscovery>
         <scopes>
            <add scope = "net.tcp://Simulation"/>
         </scopes>
      </endpointDiscovery>
   </behavior>
```

client needs at least one of the scopes to match, but not all of them.

The client has two ways of using scopes. The first is to add the scope to the finding criteria:

```
public class FindCriteria
{
   public Collection<Uri> Scopes
  {get;}
   //More members
}
```

Now the Find method will return only compatible endpoints that also list that scope. If the client adds multiple scopes, then Find will return only endpoints that support all of the listed scopes. Note that the endpoint may support additional scopes not provided to Find.

Discovery is independent of how exactly the service host defines its endpoints.

The second way of using scopes is to examine the scopes returned in FindResponse:

```
public class EndpointDiscoveryMetadata
{
   public Collection<Uri> Scopes
   {get;}
  //More members
}
```

These scopes are all the scopes supported by the endpoint, and they are useful for additional filtering.

Discovery Cardinality

Whenever relying on discovery, the client must deal with what I call *discovery cardinality*, that is, how many endpoints are discovered and which one, if any, to invoke. There are several cases of cardinality:

- No endpoint is discovered. In this case the client needs to deal with the absence of the service. This is no different from any other WCF client whose service is unavailable.
- Exactly one compatible endpoint is discovered. This is by far the most common case—the client simply proceeds to invoke the service.
- Multiple endpoints are discovered. Here the client, in theory, has two options. The first is to invoke all of them. This is the case with a publisher firing an event at subscribers, as discussed later on, and is a valid scenario. The second option is to invoke

some (including only one), but not all of the discovered endpoints). I find that scenario to be moot. Any attempt to place logic in the client that resolves which endpoint to invoke creates too much coupling across the system. It negates the very notion of runtime discovery, namely, that any discovered endpoint will do. If it is possible to discover undesirable endpoints, then using discovery is a poor design choice, and you should instead provide static addresses to the client.

If the client expects to discover exactly one endpoint (cardinality of one), then the client should instruct Find to return as soon as it finds that endpoint. Doing so will drastically reduce the discovery latency and make it adequate for the majority of cases.

The client can configure the cardinality using the MaxResults property of FindCriteria:

```
public class FindCriteria
{
   public int MaxResults
   {get;set;}
   //More members
}
FindCriteria criteria = new FindCriteria(typeof(IMyContract));
criteria.MaxResults = 1;
```

You can streamline the case of cardinality of one using my DiscoveryHelper.DiscoverAddress<T> helper method:

```
public static class DiscoveryHelper
{
   public static EndpointAddress DiscoverAddress<T>(Uri scope = null);
   //More members
}
```

Using DiscoverAddress<T>, **Figure 4** is reduced to:

```
EndpointAddress address = DiscoveryHelper.DiscoverAddress<IMyContract>();
Binding binding = new NetTcpBinding();
IMyContract proxy = ChannelFactory<IMyContract>.
CreateChannel(binding,address);
proxy.MyMethod();
(proxy as ICommunicationObject).Close();
```

Streamlining Discovery

So far, the client has had to hardcode the binding to use. However, if the service supports a MEX endpoint, the client can discover the MEX endpoint address, then proceed to retrieve and process the metadata in order to obtain the binding to use, along with the endpoint address. To help with MEX endpoint discovery, the FindCriteria class offers the static method CreateMetadataExchangeEndpointCriteria:

```
public class FindCriteria
{
   public static FindCriteria CreateMetadataExchangeEndpointCriteria();
//More members
}
```

To streamline this sequence, use my DiscoveryFactory. CreateChannel<T> method:

```
public static class DiscoveryFactory
{
   public static T CreateChannel<T>(Uri scope = null);
   //More members
}
```

Using CreateChannel<T>, **Figure 4** is reduced to:

```
IMyContract proxy = DiscoveryFactory.CreateChannel<IMyContract>();
proxy.MyMethod();
(proxy as ICommunicationObject).Close();
```

CreateChannel<T> assumes cardinality of one with the MEX endpoint (that is, only a single discoverable MEX endpoint is found in the local network), and that the metadata contains exactly one endpoint whose contract is the specified type parameter T.

Figure 7 **WCF Implementation of an Announcements Endpoint**

```
public class AnnouncementEventArgs : EventArgs
{
   public EndpointDiscoveryMetadata EndpointDiscoveryMetadata
   {get;}
   //More members
}
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,
              ConcurrencyMode = ConcurrencyMode.Multiple)]
public class AnnouncementService : ...
{
   public event EventHandler<AnnouncementEventArgs>
OfflineAnnouncementReceived;
   public event EventHandler<AnnouncementEventArgs>
OnlineAnnouncementReceived;
   //More members
}
```

Note that CreateChannel<T> uses the MEX endpoint both for the endpoint binding and address. The service is expected to support both a MEX endpoint and a discovery endpoint (although the client never uses the discovery endpoint to find the actual endpoint).

In case there are multiple services supporting the desired service contract, or there are multiple MEX endpoints, DiscoveryFactory also offers the CreateChannels<T> method:

```
public static class DiscoveryHelper
{
   public static T[] CreateChannels<T>(bool inferBinding = true);
   //More members
}
```

CreateChannels<T> by default will infer the binding to use from the scheme of the service endpoint. If inferBinding is false, it will discover the binding from the MEX endpoints.

CreateChannels<T> does not assume a cardinality of one on the compatible service endpoints or the MEX endpoints, and will return an array of all compatible endpoints.

Announcements

The discovery mechanism as presented thus far is passive from the perspective of the service. The client queries the discovery endpoint and the service responds. As an alternative to this passive address discovery, WCF offers an active model, where the service broadcasts its status to all clients and provides its address. The service host broadcasts a "hello"

Figure 8 **Using AnnouncementSink<T>**

```
class MyClient : IDisposable
{
   AnnouncementSink<IMyContract> m_AnnouncementSink;
   public MyClient()
   {
      m_AnnouncementSink = new AnnouncementSink<IMyContract>();
      m_AnnouncementSink.OnHelloEvent += OnHello;
      m_AnnouncementSink.Open();
   }
   void Dispose()
   {
      m_AnnouncementSink.Close();
   }
   void OnHello(string address)
   {
      EndpointAddress endpointAddress = new EndpointAddress(address);
   IMyContract proxy = ChannelFactory<IMyContract>.CreateChannel(
      new NetTcpBinding(),endpointAddress);
      proxy.MyMethod();
      (proxy as ICommunicationObject).Close();
   }
}
```

announcement when the host is opened and a "bye" announcement when the host shuts down gracefully. If the host is aborted ungracefully, no "bye" announcement is sent. These announcements are received on a special announcements endpoint hosted by the client (see **Figure 6**).

Announcements are an individual endpoint-level mechanism, not a host-level one. The host can choose which endpoint to announce. Each announcement contains the endpoint address, its scopes and its contract.

Discovery is geared toward addresses only. There is no information about which binding to use to invoke the service.

Note that announcements are unrelated to address discovery. The host may not support a discovery endpoint at all, and there is no need for the discovery behavior. On the other hand, the host may chose to both support the discovery endpoint and announce its endpoints, as shown in **Figure 6**.

The host can automatically announce its endpoints. All you need to do is provide the information about the client announcement endpoint for the discovery behavior. For example, when using a config file:

```
<behavior>
   <serviceDiscovery>
      <announcementEndpoints>
         <endpoint
            kind = "udpAnnouncementEndpoint"
         />
      </announcementEndpoints>
   </serviceDiscovery>
</behavior>
```

My EnableDiscovery extension method also adds the announcement endpoint to the discovery behavior.

For the use of the client, WCF provides a pre-canned implementation of an announcements endpoint with the AnnouncementService class, as shown in **Figure 7**.

AnnouncementService is a singleton configured for concurrent access. AnnouncementService provides two event delegates that the client can subscribe to in order to receive the announcements. The client should host the AnnouncementService using the constructor of ServiceHost, which accepts a singleton instance. This is required so that the client is able to interact with the instance and subscribe to the events. In addition, the client must add the UDP announcement endpoint to the host:

```
AnnouncementService announcementService = new AnnouncementService();
announcementService.OnlineAnnouncementReceived  += OnHello;
announcementService.OfflineAnnouncementReceived += OnBye;
ServiceHost host = new ServiceHost(announcementService);
host.AddServiceEndpoint(new UdpAnnouncementEndpoint());
host.Open();
void OnHello(object sender,AnnouncementEventArgs args)
{...}
void OnBye(object sender,AnnouncementEventArgs args)
{...}
```

There is one important detail related to receiving announcements. The client would receive all notifications of all services in the intranet, regardless of contract type or, for that matter, applications or scopes. The client must filter out the relevant announcements.

Streamlining Announcements

You can greatly simplify and improve on the raw steps required of the client to utilize announcements using my AnnouncementSink<T> class defined as:

```
public class AnnouncementSink<T> : AddressesContainer<T> where T: class
{
   public event Action<T> OnHelloEvent;
   public event Action<T> OnByeEvent;
}
```

AnnouncementSink<T> automates hosting the announcements endpoint by encapsulating the steps of **Figure 7**. While AnnouncementSink<T> hosts an instance of AnnouncementService internally, it improves on its deficiencies. First, AnnouncementSink<T> offers two event delegates for notifications. Unlike the raw AnnouncementService, AnnouncementSink<T> fires these delegates concurrently. In addition, AnnouncementSink<T> disables the synchronization context affinity of AnnouncementService, so that it can accept the announcements on any incoming thread, making it truly concurrent.

AnnouncementSink<T> filters the contract types and only fires its events when compatible endpoints announce themselves. The only thing the client needs to do is to open and close AnnouncementSink<T>, in order to indicate when to start and stop receiving notifications.

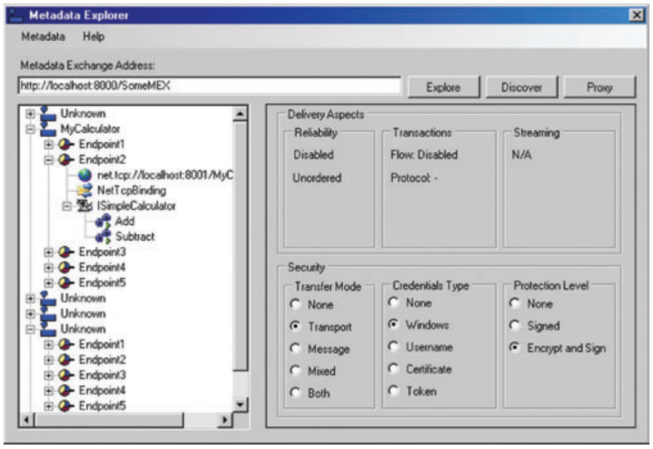
AnnouncementSink<T> derives my general-purpose address container called AddressesContainer<T>.

AddressesContainer<T> is a rich address-management helper collection that you can use whenever you need to manipulate multiple addresses. AddressesContainer<T> supports several iterators, indexers, conversion methods and queries.

Figure 8 demonstrates using AnnouncementSink<T>.

The MEX Explorer

In my book "Programming WCF Services Second Edition" (O'Reilly, 2008), I presented a tool I call the MEX Explorer (see **Figure 9**). You can provide a MEX address to the MEX Explorer and use it to reflect the service endpoints (their address, binding properties and contract). The introduction of discovery enabled me to revamp the MEX Explorer.



Metadata Exchange Address: http://localhost:8000/SomeMEX

Unknown
MyCalculator
Endpoint1
Endpoint2
net.tcp://localhost:8001/MyC
NetTcpBinding
ISimpleCalculator
Add
Subtract
Endpoint3
Endpoint4
Endpoint5
Unknown
Unknown
Unknown
Endpoint1
Endpoint2
Endpoint3
Endpoint4
Endpoint5

Reliability: Disabled, Unordered
Transactions: Flow: Disabled, Protocol: -
Streaming: N/A

Transfer Mode: None [✓ Transport] Message Mixed Both
Credentials Type: None [✓ Windows] Username Certificate Token
Protection Level: None Signed [✓ Encrypt and Sign]

Figure 9 **The MEX Explorer**

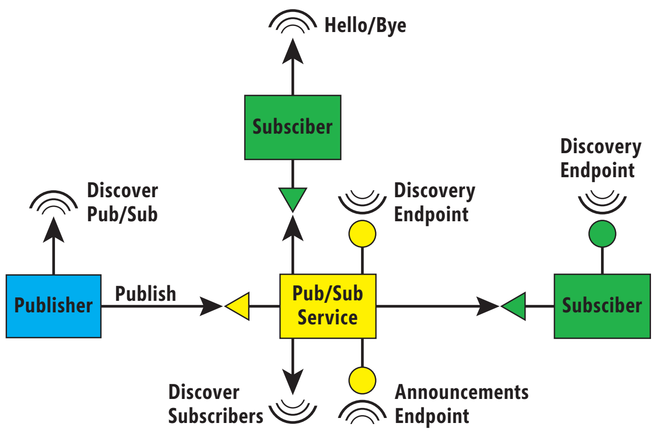


Figure 10 **Discovery-Driven Publish-Subscribe System**

Clicking the Discover button triggers a discovery request for all MEX endpoints without any limit on cardinality. The tool then visualizes all discovered endpoints in the tree. In addition, The MEX Explorer utilizes announcements of MEX endpoints. In responding to the announcements, the MEX Explorer refreshes itself and presents the new endpoints or removes from the tree those that are no longer running.

Discovery-Driven Publish-Subscribe Pattern

In the October 2006 article, "What You Need To Know About One-Way Calls, Callbacks and Events" (msdn.microsoft.com/magazine/cc163537), I present my framework for supporting a publish-subscribe pattern in WCF. You can use the mechanisms of discovery and announcements to provide yet another way of implementing a publish-subscribe system.

Unlike the techniques in that article, a discovery-based solution is the only publish-subscribe case that requires no explicit steps by the subscribers or administrator. When utilizing discovery, there is no need to explicitly subscribe either in code or in config. In turn, this significantly simplifies the deployment of the system, and it enables great flexibility in the presence of both publishers and subscribers. You can easily add or remove subscribers and publishers without any additional administration steps or programming.

When taking advantage of discovery for a publish-subscribe system, the subscribers can provide a discovery endpoint so that the publish-subscribe service can discover them, or they can announce their event-handling endpoints, or even do both.

The publishers should not discover the subscribers directly, because that may incur the discovery latency on every event firing (having the cardinality of all endpoints). Instead, the publishers should discover the publish-subscribe service, which is a one-time negligible cost. The publish-subscribe service should be a singleton (enabling fast discovery since it has cardinality of one). The publish-subscribe service exposes the same event endpoint as the subscribers, so it looks like a meta-subscriber to the publishers. That is, it requires the same code to fire the event at the publish-subscribe service as against an actual subscriber.

Figure 11 **Implementing a Publish-Subscribe Service**

```
class MyPublishService : DiscoveryPublishService<IMyEvents>,IMyEvents
{
   public void OnEvent1()
   {
      FireEvent();
   }
   public void OnEvent2(int number)
   {
      FireEvent(number);
   }
}
```

The events endpoint of the publish-subscribe service must use a particular scope. This scope enables the publishers to find the publish-subscribe service rather than the subscribers. In addition to supporting discovering that scoped events endpoint, the publish-subscribe service provides an announcement endpoint.

The publish-subscribe service maintains a list of all subscribers. The publish-subscribe service can keep that list current by constantly trying to discover the subscribers using some ongoing background activity. Note again that having the publish-subscribe service's events endpoint associated with a special scope will also prevent the publish-subscribe service from discovering itself when discovering all events endpoints. The publish-subscribe service can also provide an announcement endpoint to monitor subscribers. **Figure 10** depicts this architecture.

The Publish-Subscribe Service

To facilitate deploying your own publish-subscribe service, I wrote the DiscoveryPublishService<T> defined as:

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
public class DiscoveryPublishService<T> : IDisposable where T: class
{
   public static readonly Uri Scope;
   protected void FireEvent(params object[] args);
   //More members
}
```

All you need to do is to derive your publish-subscribe service from DiscoveryPublishService<T> and specify the events contract as the type parameter. Then implement the operations of the event contract by calling the FireEvent method.

For example, consider this events contract:

```
[ServiceContract]
interface IMyEvents
{
   [OperationContract(IsOneWay = true)]
   void OnEvent1();
   [OperationContract(IsOneWay = true)]
   void OnEvent2(int number);
}
```

Figure 11 shows how to implement your publish-subscribe service using DiscoveryPublishService<T>.

Internally, DiscoveryPublishService<T> uses another one of my AddressContainer<T> derived classes called DiscoveredServices<T>, defined as:

```
public class DiscoveredServices<T> : AddressesContainer<T> where T:
class
{
   public DiscoveredServices();
   public void Abort();
}
```

DiscoveredServices<T> is designed to maintain as much as possible an up-to-date list of all discovered services, and it stores the addresses it discovers in its base class. DiscoveredServices<T> spins off an ongoing discovery on a background thread, and

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.

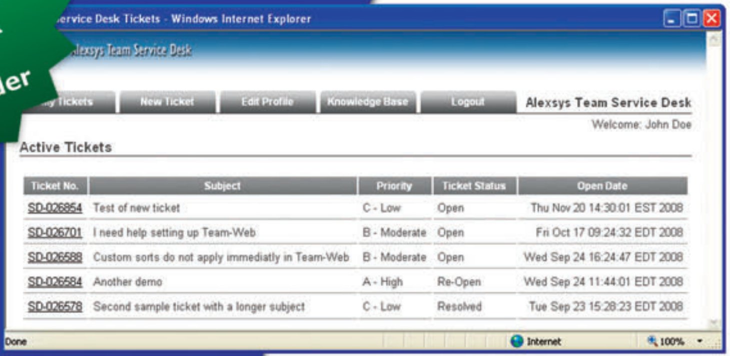
- Quality Control / Compliance Tracking
- Project Management
- Bugs and Features
- Help Desk
- Sales and Marketing
- Action Items

New Alexsys Team Service Desk with Visual Builder

Alexsys Team

New in Team 2.10

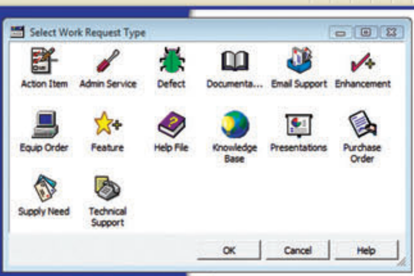
- Calculated Fields, Formulas and Functions
- Field display formats
- XML and .NET support
- Database Migration tool
- Attachment and URL Links



ervice Desk Tickets - Windows Internet Explorer
Alexsys Team Service Desk
My Tickets
New Ticket
Edit Profile
Knowledge Base
Logout
Alexsys Team Service Desk
Welcome: John Doe
Done
Internet
100%

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



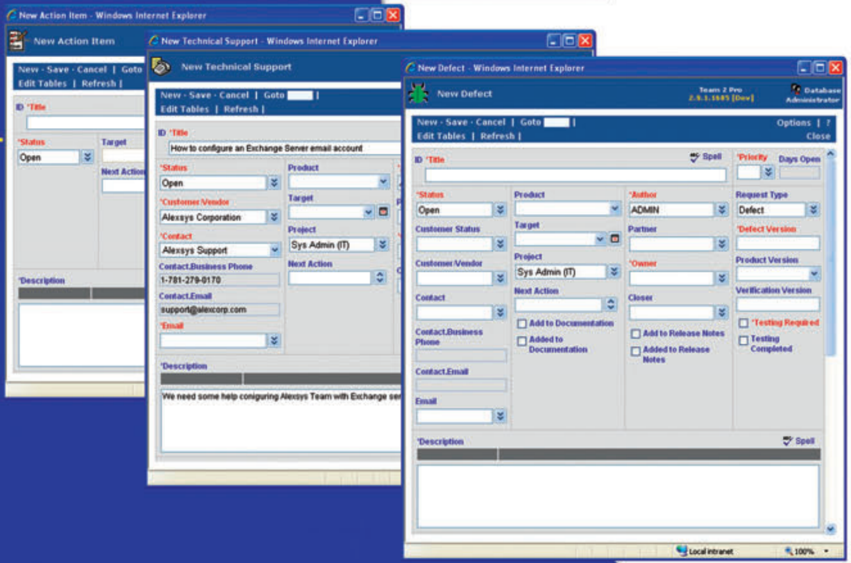
New Service Desk

- 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
New Action Item
New · Save · Cancel | Goto
Edit Tables | Refresh |
New Technical Support - Windows Internet Explorer
New Technical Support
New Defect - Windows Internet Explorer
New Defect
Team 2 Pro 2.9.1.1685 [Dev]
Database Administrator
Options | ?
Close
Spell
Local intranet
100%

New Action Item: ID *Title; *Status: Open; Target; Next Action; *Description

New Technical Support: ID *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; *Email; Product; Target; Project: Sys Admin (IT); Next Action; *Description: We need some help coniguring Alexsys Team with Exchange se

New Defect: ID *Title; *Priority; Days Open; *Status: Open; Customer Status; Customer/Vendor; Contact; Contact.Business Phone; Contact.Email; Email; Product; Target; Project: Sys Admin (IT); Next Action; ☐ Add to Documentation; ☐ Added to Documentation; *Author: ADMIN; Partner; *Owner; Closer; ☐ Add to Release Notes; ☐ Added to Release Notes; Request Type: Defect; *Defect Version; Product Version; Verification Version; ☐ *Testing Required; ☐ Testing Completed; *Description

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.

Discover
SCALABLE
Performance

SCALEOUT STATESERVER®
Simply Powerful Distributed Caching

As a software architect, you know the importance of combining simplicity with power. ScaleOut StateServer's distributed cache gives you both.

Its powerful architecture simplifies both configuration and management so that you can quickly - and easily - tap into blazing performance, scalability, and 24x7 availability.

Give your server farm and grid applications the performance boost they need. Download your FREE trial copy of ScaleOut StateServer today!

SCALEOUT SOFTWARE
www.scaleoutsoftware.com/eval | (503) 643-3422

it is useful in cases where you want a current repository of discovered addresses.

The FireEvent method extracts from the message headers the operation name. It then queries the subscribers list for all subscribers that do not support the publish-subscribe scope (to avoid self discovery). FireEvent then merges the lists into a union of unique entries (this is required to deal with subscribers that both announce themselves and are discoverable). For each subscriber, FireEvent infers the binding from the address scheme and creates a proxy to fire at the subscriber. Publishing the events is done concurrently using threads from the thread pool.

A discovery-based solution is the only publish-subscribe case that requires no explicit steps by the subscribers or administrator.

To host your publish-subscribe service, use the static helper method CreateHost<S> of DiscoveryPublishService<T>:

```
public class DiscoveryPublishService<T> : IDisposable where T: class
{
   public static ServiceHost<S> CreateHost<S>()
     where S : DiscoveryPublishService<T>,T;
   //More members
}
```

The type parameter S is your subclass of DiscoveryPublishService<T>, and T is the events contract. CreateHost<S> returns an instance of a service host you need to open:

```
ServiceHost host = DiscoveryPublishService<IMyEvents>.
  CreateHost<MyPublishService>();
host.Open();
```

In addition, CreateHost<S> will also obtain an available TCP base address and add the events endpoint so there is no need for a config file.

The Publisher

The publisher needs a proxy to the events service. For that, use my DiscoveryPublishService<T>.CreateChannel:

```
public class DiscoveryPublishService<T> : IDisposable where T : class
{
   public static T CreateChannel();
   //More members
}
```

DiscoveryPublishService<T>.CreateChannel discovers the publish-subscribe service and creates a proxy to it. That discovery is fast since the cardinality is one. The code of the publisher is straightforward:

```
IMyEvents proxy = DiscoveryPublishService<IMyEvents>.CreateChannel();
proxy.OnEvent1();
(proxy as ICommunicationObject).Close();
```

When it comes to implementing a subscriber, there is nothing special to do. Simply support the events contract on a service, and add either discovery or announcements (or both) of the events endpoint. ■

JUVAL LOWY *is a software architect with IDesign and provides WCF training and architecture consulting. His latest book is "Programming WCF Services Third Edition" (O'Reilly, 2010). He is also the Microsoft regional director for the Silicon Valley. Contact Lowy at www.idesign.net.*

Web Application Request-Response Testing with JavaScript

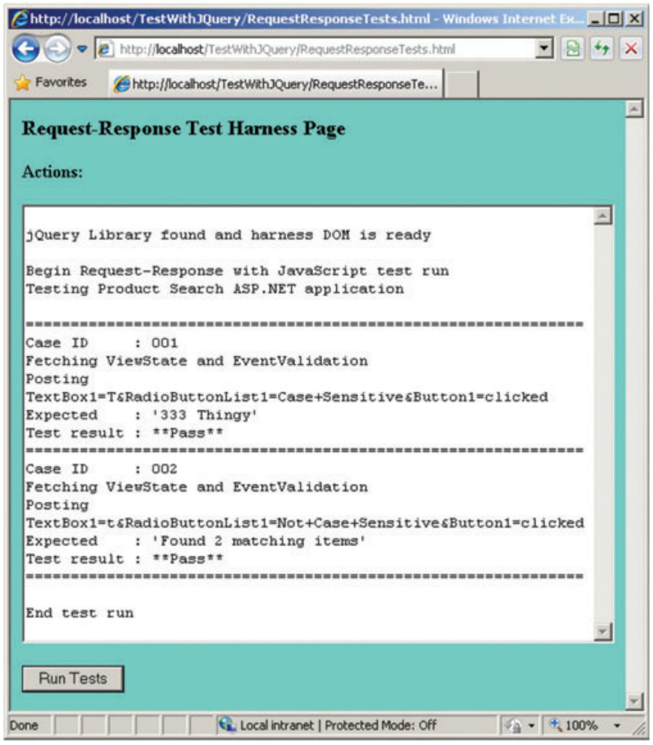
In this month's column I explain how to write simple and effective browser-based request-response test automation using JavaScript. The best way for you to see where I'm headed is to take a look at the screenshots in **Figures 1** and **2**. **Figure 1** shows a simple but representative ASP.NET Web application under test named Product Search. A user enters some search string into the application's single textbox control, and specifies whether the search is to be performed in a case-sensitive manner using two radio button controls. Search results are displayed in a listbox control.

Although the example Web application under test is based on ASP.NET, the technique I present in this article can be used to create test automation for Web applications written using most dynamic page generation technologies, including PHP, JSP, CGI and others.

Figure 2 shows the request-response test automation in action. Notice that the test automation harness is browser-based. One of the advantages of the technique I present here compared to alternative approaches is that the technique can be used with most major Web browsers, and can be executed on test host machines running most operating systems.

The test automation harness consists of a single HTML page that houses a relatively short set of JavaScript functions. Notice that the first line of test run output indicates that the test automation is using the jQuery library. The harness reads test case input data, which corresponds to user input, and programmatically posts that input data to the Product Search Web application. The harness accepts the resulting HTTP response data and examines that response for an expected value in order to determine a test case pass/fail result.

In the sections of this article that follow, I first briefly describe


http://localhost/TestWithJQuery/RequestResponseTests.html - Windows Internet Ex...
http://localhost/TestWithJQuery/RequestResponseTests.html
Favorites
http://localhost/TestWithJQuery/RequestResponseTe...
Run Tests
Done
Local intranet | Protected Mode: Off
100%

Request-Response Test Harness Page

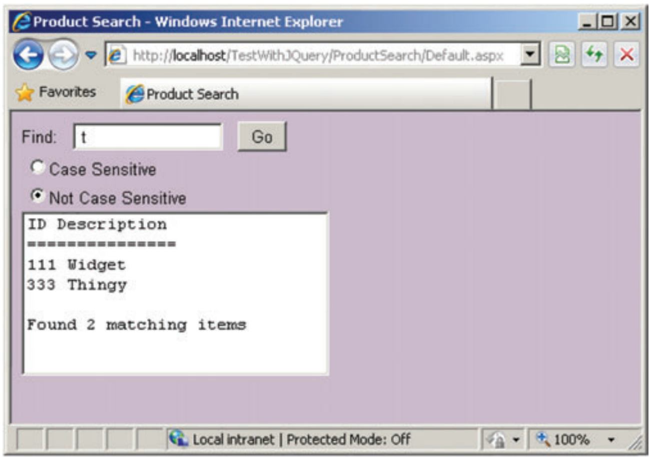
Actions:

```
jQuery Library found and harness DOM is ready

Begin Request-Response with JavaScript test run
Testing Product Search ASP.NET application

==============================================================
Case ID     : 001
Fetching ViewState and EventValidation
Posting
TextBox1=T&RadioButtonList1=Case+Sensitive&Button1=clicked
Expected    : '333 Thingy'
Test result : **Pass**
==============================================================
Case ID     : 002
Fetching ViewState and EventValidation
Posting
TextBox1=t&RadioButtonList1=Not+Case+Sensitive&Button1=clicked
Expected    : 'Found 2 matching items'
Test result : **Pass**
==============================================================

End test run
```

Figure 2 **Request-Response Test Run**

the Web application under test shown in Figure 1 so that you'll understand which factors are relevant to HTTP request-response testing. Next I explain in detail the test harness code shown running in **Figure 2** so that you'll be able to modify the harness to meet your own needs. I conclude with a few comments about when browser-based request-response test automation with JavaScript is appropriate and when alternative approaches might be more suitable.

This article assumes you have intermediate level JavaScript and ASP.NET skills, but even if you're a beginner with these technologies you should be able to follow my explanations without too much difficulty.


Product Search - Windows Internet Explorer
http://localhost/TestWithJQuery/ProductSearch/Default.aspx
Favorites
Product Search
Go
Local intranet | Protected Mode: Off
100%

Find: t

Case Sensitive

[✓ Not Case Sensitive]

```
ID Description
===============
111 Widget
333 Thingy

Found 2 matching items
```

Figure 1 **Product Search Web Application Under Test**

Building the Web Application

I used Visual Studio 2008 to create the Product Search Web application under test. In order to leverage Visual Studio's

Send your questions and comments for Dr. McCaffrey to testrun@microsoft.com.

Code download available at code.msdn.microsoft.com/mag2010Test.

ability to configure a Web site, I selected File | New | Web Site from the main menu bar. Next I selected the Empty Web Site option from the resulting new Web site dialog box. I specified an HTTP location on my local machine to create a complete ASP.NET Web site, rather than specifying a file system location to use the built-in Visual Studio development server. I selected the C# language for my logic code.

After clicking OK, Visual Studio created the empty ProductSearch Web site. In the Solution Explorer window, I right-clicked on the ProductSearch project and selected Add New Item from the context menu. I selected the Web Form item and accepted the default page name of Default.aspx and clicked Add to generate the page. Next I created the simple UI for the Web application under test, as presented in **Figure 3**.

As I will explain shortly, when creating HTTP request-response test automation you must know the IDs of any of the input controls that you wish to simulate user input on. In this case I have access to the source code of the application under test, but even if you do not have source code access you can always determine input control IDs using a Web browser's view-source functionality. Notice that the two radio button controls are actually represented by a single input control with ID RadioButtonList1, rather than by two controls as you might have guessed.

You can always determine input control IDs using a Web browser's view-source functionality.

I added the application logic directly into the Defaut.aspx file rather than using the code-behind mechanism. At the top of the page I created a script block to hold the application's logic code:

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
  // ...
</script>
```

I added a small class into the script block to represent a Product object:

```
public class Product {
  public int id;
  public string desc;
  public Product(int id, string desc) {
    this.id = id; this.desc = desc;
  }
}
```

Then I added an internal application-scope ArrayList object to simulate an external data store:

```
public static ArrayList data = null;
```

In most realistic Web application scenarios, data stores are usually external, such as an XML file or SQL Server database. However, when performing HTTP request-response testing, the location of an application's data store is irrelevant to some extent. The HTTP request has no knowledge of the data store's location, and the HTTP response typically contains only HTML. Next I added some code to populate the internal data store with Product items:

```
protected void Page_Load(object sender, EventArgs e) {
  if (!IsPostBack) {
    data = new ArrayList();
    Product p1 = new Product(111, "Widget");
    Product p2 = new Product(222, "Gizzmo");
    Product p3 = new Product(333, "Thingy");
    data.Add(p1); data.Add(p2); data.Add(p3);
  }
}
```

Finally, I placed all the application logic into the event handler for the Button1 click event. I begin by clearing the ListBox1 result area and fetching user input:

```
ListBox1.Items.Clear();
string filter = TextBox1.Text.Trim();
string sensitivity = RadioButtonList1.SelectedValue;
```

The sensitivity string variable will hold either "Case Sensitive" or "Not Case Sensitive."

Next I place header information into the ListBox1 result area and declare a string to hold a Product search result and initialize a counter to track how many Product items match the search filter:

```
ListBox1.Items.Add("ID   Description");
ListBox1.Items.Add("================");
string resultRow;
int count = 0;
```

I iterate through each Product object in the ArrayList data store checking to see if the search filter string matches the current object's description field:

```
foreach (Product p in data) {
  resultRow = "";
  if (sensitivity == "Not Case Sensitive" &&
    p.desc.IndexOf(filter,
    StringComparison.CurrentCultureIgnoreCase) >= 0) {
    resultRow = p.id + " " + p.desc; ++count;
  }
  else if (sensitivity == "Case Sensitive" &&
    p.desc.IndexOf(filter) >= 0) {
    resultRow = p.id + " " + p.desc; ++count;
  }
  if (resultRow != "") ListBox1.Items.Add(resultRow);
}
```

Figure 3 Web App UI

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Product Search</title>
</head>
<body bgcolor="#ccbbcc">
  <form id="form1" runat="server">
  <div>
    <asp:Label ID="Label1" runat="server" Text="Find:"
      Font-Names="Arial" Font-Size="Small">
    </asp:Label>&nbsp;&nbsp;
    <asp:TextBox ID="TextBox1" runat="server" Width="114px">
    </asp:TextBox>&nbsp;&nbsp;
    <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
      Text="Go" />
    <br />

    <asp:RadioButtonList ID="RadioButtonList1" runat="server"
      Font-Names="Arial" Font-Size="Small">
    <asp:ListItem>Case Sensitive</asp:ListItem>
    <asp:ListItem Selected="True">Not Case Sensitive</asp:ListItem>
    </asp:RadioButtonList>
  </div>
  <asp:ListBox ID="ListBox1" runat="server" Height="131px" Width="246px"
    Font-Names="Courier New" Font-Size="Small">
  </asp:ListBox>
  </form>
</body>
</html>
```

Figure 4 **Test Harness Structure**

```
<html>
<!-- RequestResponseTests.html -->
<head>
  <script src='http://localhost/TestWithJQuery/jquery-1.3.2.js'>
  </script>
  <script type="text/javascript">
    $(document).ready(function() {
      logRemark("jQuery Library found and harness DOM is ready\n");
    } );

    var targetURL = 'http://localhost/TestWithJQuery/ProductSearch/
Default.aspx';

    var testCaseData =
[ '001,TextBox1=T&RadioButtonList1=Case+Sensitive&Button1=clicked,333
Thingy',
'002,TextBox1=t&RadioButtonList1=Not+Case+Sensitive&Button1=clicked,Found 2
matching items' ];

    function runTests() {
      try {
        logRemark('Begin Request-Response with JavaScript test run');
        logRemark("Testing Product Search ASP.NET application\n");
        // ...
        logRemark("\nEnd test run");
      }
      catch(ex) {
        logRemark("Fatal error: " + ex);
      }
    }

    function getVS(target) {
      // ...
    }

    function getEV(target) {
      // ...
    }

    function sendAndReceive(target, rawVS, rawEV, inputData) {
      // ...
    }

    function logRemark(comment) {
      // ...
    }

  </script>
</head>
<body bgcolor="#66ddcc">
  <h3>Request-Response Test Harness Page</h3>
  <p><b>Actions:</b></p><p>
  <textarea id="comments" rows="24" cols=63">
  </textarea></p>
  <input type="button" value="Run Tests" onclick="runTests();" />
</body>
</html>
```

For each product that matches the search filter, I build a result string and increment the hit counter. Notice that the IndexOf method is conveniently overloaded to accept a case-sensitivity argument.

The application logic finishes by adding a blank line and a count summary to the ListBox1 display area:

```
ListBox1.Items.Add("");
ListBox1.Items.Add("Found " + count + " matching items");
```

In order to keep the size of the Web application as small and simple as possible, I have taken many shortcuts you wouldn't use in a production environment. In particular I have not provided any error checking or handling.

Request-Response Test Automation

I created the test harness page shown running in **Figure 2** using notepad. The overall structure of the harness is shown in **Figure 4**.

The harness UI code in the body element at the bottom of the page consists only of some text, a textarea element to display information and a button to start the test automation.

The test harness structure begins by using the script element src attribute to reference the jQuery library. The jQuery library is an open source collection of JavaScript functions available from jquery.com. Although jQuery was created with Web development in mind, the library contains functions that make it well suited for lightweight request-response test automation. Here I point to a local copy of version 1.3.2 of the library. For test-automation purposes, using a local copy of the library is more reliable than pointing to a remote copy. Next I use the $(document).ready jQuery idiom to make sure that my harness can access the library and that the harness DOM is loaded into memory.

After setting up a variable targetURL that points to the Web application under test, I hard code internal comma-delimited test cases into a string array named testCaseData. Here I have just two test cases, but in a production environment you might have hundreds of cases. External test case data is often preferable to internal test case data because external data can be more easily modified and shared. However, because the technique I'm presenting here is lightweight, internal test case data is a reasonable design choice.

The first field in a test case is a case ID number. The second field is raw request data to send to the application under test. The third field is an expected result.

How did I know the format of the request data? The easiest way to determine the format of HTTP request data is to perform preliminary experimentation with the application under test by examining actual request data using an HTTP logger tool such as Fiddler.

Running the Tests

The main harness control function is named runTests. The runTests function uses a top-level try-catch mechanism to provide rudimentary error handling. I use an auxiliary function named logRemarks to display information to the harness textarea element. The harness uses helper functions getVS and getEV to get the current ViewState and EventValidation values of the ASP.NET Web application under test. These application-generated Base64-encoded values act primarily as state and security mechanisms, and must be sent as part of any HTTP POST request. The sendAndReceive function performs the actual HTTP request and returns the corresponding HTTP response:

```
The runTests function iterates through each test case:
for (i = 0; i < testCaseData.length; ++i) {
  logRemark("==========================");
  var tokens = testCaseData[i].split(',');
  var caseID = tokens[0];
  var inputData = tokens[1];
  var expected = tokens[2];
  ...
```

I use the built-in split function to separate each test case string into smaller pieces. Next I call the getVS and getEV helper functions:

```
logRemark('Case ID     : ' + caseID);
logRemark('Fetching ViewState and EventValidation');
var rawVS = getVS(targetURL);
var rawEV = getEV(targetURL);
```

The main processing loop continues by calling the sendAndReceive function and examining the resulting HTTP response for the associated test case expected value:

```
var response = sendAndReceive(targetURL, rawVS, rawEV, inputData);
logRemark("Expected    : '" + expected + "'");
if (response.indexOf(expected) >= 0)
  logRemark("Test result : **Pass**");
else if (response.indexOf(expected) == -1)
  logRemark("Test result : **FAIL**");
} // main loop
```

The getVS helper function relies on the jQuery library:

```
function getVS(target) {
  $.ajax({
    async: false, type: "GET", url: target,
    success: function(resp) {
      if (resp.hasOwnProperty("d")) s = resp.d;
      else s = resp;

      start = s.indexOf('id="__VIEWSTATE"', 0) + 24;
      end = s.indexOf('"', start);
    }
  });
  return s.substring(start, end);
}
```

The main idea of the getVS function is to send a priming GET request to the application under test, fetch the response and parse out the ViewState value. The $.ajax function accepts an anonymous function. The async, type and URL parameters should be fairly self-explanatory. The hasOwnProperty("d") method of the response resp object is essentially a security mechanism present in the Microsoft .NET Framework 3.5 and is not necessary in this situation.

I use jQuery selector and chaining syntax to get the current text in the textarea element.

I extract the ViewState value by looking for the start of the attribute, then counting over 24 characters to where the ViewState value actually begins. The getEV function code is exactly the same as the getVS code except that the EventValidation value starts 30 characters from the initial id=EVENTVALIDATION attribute. Having separate getVS and getEV functions gives you flexibility but requires two separate priming requests. An alternative is to refactor getVS and getEV into a single helper function.

The sendAndReceive helper function executes the actual HTTP request and fetches the resulting response. The function begins by converting the raw ViewState and EventValidation strings into URL-encoded strings, and then constructs the data to post to the Web application:

```
function sendAndReceive(target, rawVS, rawEV, inputData) {
  vs = encodeURIComponent(rawVS);
  ev = encodeURIComponent(rawEV);
  postData = inputData + '&__VIEWSTATE=' + vs +
    '&__EVENTVALIDATION=' + ev;
  ...
```

The built-in encodeURIComponent function encodes characters that are not legal values in post data into an escape sequence. For example, the '/' character is encoded as %2F. After a logging message, sendAndReceive uses the $.ajax method to create an HTTP POST request:

```
logRemark("Posting " + inputData);
$.ajax({
  async: false,
  type: "POST",
  url: target,
  contentType: "application/x-www-form-urlencoded",
  data: postData,
  ...
```

The $.ajax method was created primarily to send asynchronous XML HTTP requests, but by setting the async parameter to false the method can be used to send standard synchronous requests. Neat! You can think of the content-type parameter value as a magic string that simply means data posted from an HTML form element. The sendAndReceive function uses the same pattern as getVS to grab the associated HTTP response:

```
    success: function(resp, status) {
      if (resp.hasOwnProperty("d")) s = resp.d;
      else s = resp;
    },
    error: function(xhr, status, errObj) {
      alert(xhr.responseText);
    }
  });
  return s;
}
```

I also use the optional error parameter to display any fatal errors in an alert box.

The final function in the test harness is the logRemark utility:

```
function logRemark(comment) {
  var currComment = $("#comments").val();
  var newComment = currComment + "\n" + comment;
  $("#comments").val(newComment);
}
```

I use jQuery selector and chaining syntax to get the current text in the textarea element, which has an ID of comments. The '#' syntax is used to select an HTML element by ID, and the val function can act as both a value setter and getter. I append the comment parameter value and a newline character to the existing comment text, and then use jQuery syntax to update the textarea element.

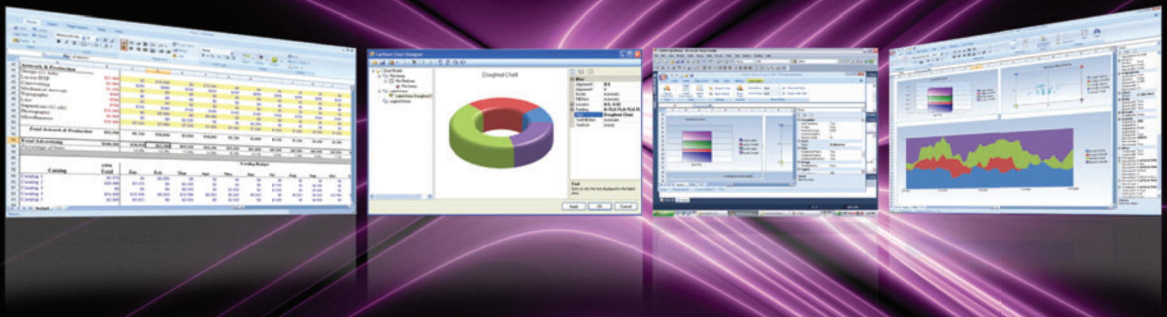
Alternatives

The main alternative to the browser-based, JavaScript language approach I've presented in this article is to create a shell-based harness using a language such as C#. Compared to a shell-based approach, the browser-based approach is most useful when you're working in a highly dynamic environment where your test automation has a short lifespan. Additionally, the browser-based approach presented here is quite platform-independent. The technique will work with any browser and OS combination that supports the jQuery library and JavaScript. ■

Dr. James McCaffrey *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at Microsoft. He has worked on several Microsoft products including Internet Explorer and Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006) and can be reached at jmccaffrey@volt.com or v-jammc@microsoft.com.*

Today, over 100,000 developers are using FarPoint Spread components.

Spread for Windows Forms 5 provides 100's of chart styles.



That is potentially over 10 million Spreadsheet Dashboards!

Don't Get Left Behind!

See how to build your dashboard at FarPointSpread.com/Dashboard/

World's #1 Microsoft Excel Compatible Spreadsheet Components

Demand the best. When you need high-performance, fully-customizable spreadsheet components, join the professional developers around the world who consistently turn to FarPoint Spread for powerful, extendable spreadsheet solutions. Download your free evaluation today.

New! Hundreds of Charts Styles to visualize your data
New! Advanced Microsoft Excel Compatibility Features
New! Spread Designer enhancements/Quick Start Wizard for rapid development
New! Additional Functions to perform advanced calculations
New! Enhanced User Interface attributes for even more control

"Ask me about the new SITE LICENSE!"

New Release

RELEASE 5.0 SPREAD for Windows Forms

GrapeCity®

www.FarPointSpread.com

Complete Business Intelligence Tools for .NET

Spreadsheets

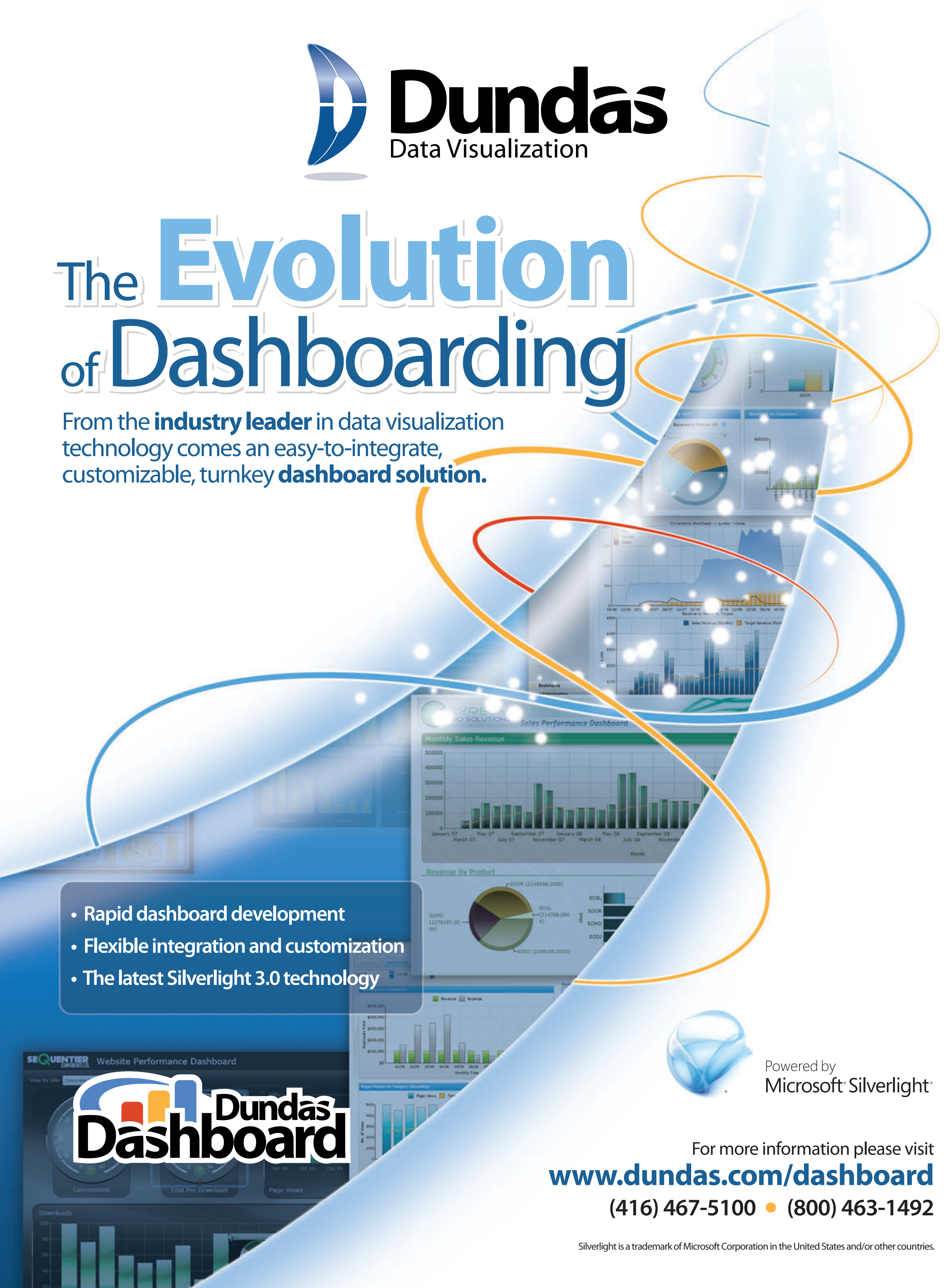
NEW Reporting

Analysis

Dundas
Data Visualization

The Evolution of Dashboarding

From the **industry leader** in data visualization technology comes an easy-to-integrate, customizable, turnkey **dashboard solution.**



- Rapid dashboard development
- Flexible integration and customization
- The latest Silverlight 3.0 technology

Dundas Dashboard

Powered by Microsoft Silverlight

For more information please visit
www.dundas.com/dashboard
(416) 467-5100 • (800) 463-1492