

magazine msdn®

Advanced Programming Made Easy with Visual Studio LightSwitch

Beth Massi 26

Build Business Applications with Visual Studio LightSwitch

Robert Green 40

Building Apps with HTML5: What You Need to Know

Brandon Satrom 52

Build a Ticketing System Using Exchange and Team Foundation Server

Mohammad Jalloul 58

The Past, Present and Future of Parallelizing .NET Applications

Stephen Toub 72

Portable Class Libraries: A Primer

Bill Kratochvil 78

Particle Swarm Optimization

James McCaffrey 86

COLUMNS

CUTTING EDGE

Static Code Analysis and Code Contracts
Dino Esposito page 6

WINDOWS WITH C++

The Windows Thread Pool and Work
Kenny Kerr page 12

FORECAST: CLOUDY

Searching Windows Azure Storage with Lucene.Net
Joseph Fultz page 16

UI FRONTIERS

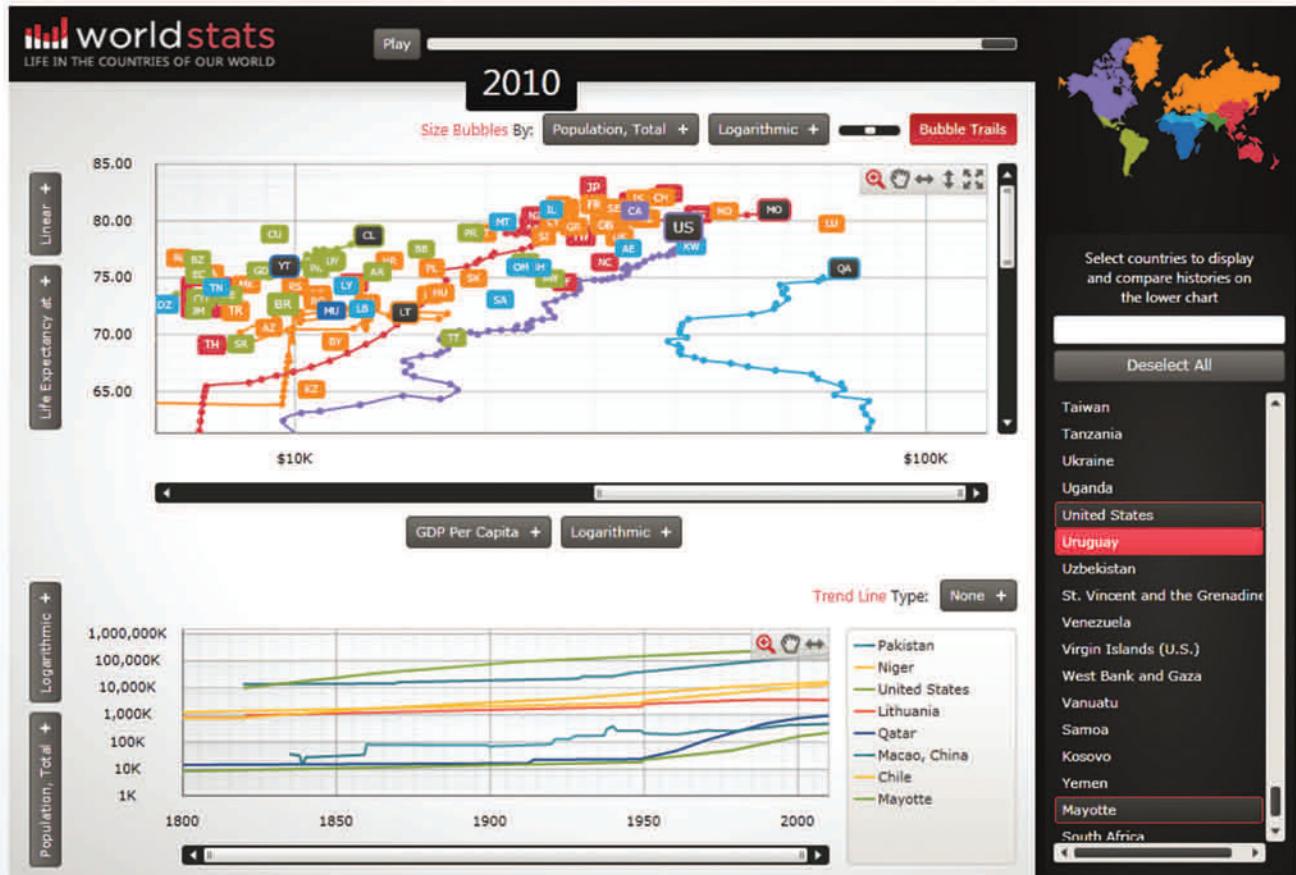
Font Metrics in Silverlight
Charles Petzold page 92

DON'T GET ME STARTED

The Power of the Default
David Platt page 96

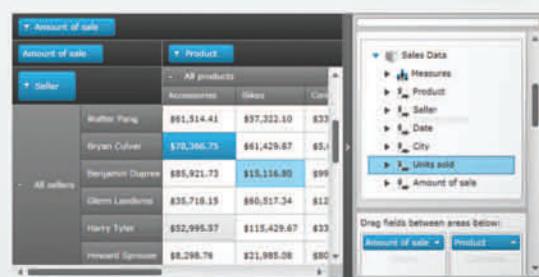
NetAdvantage[®] ULTIMATE

REPORTING, DATA VISUALIZATION AND LOB UI CONTROLS FOR ASP.NET, WINDOWS FORMS, JQUERY/HTML5, WPF, SILVERLIGHT AND WINDOWS PHONE 7



INFRASTICS MOTION FRAMEWORK™

Delivering a great user experience in Windows Presentation Foundation (WPF) and Microsoft Silverlight business intelligence applications requires more than styling, it requires giving your application's end users greater insight into the story of their data.



OLAP PIVOT GRID DATA VISUALIZATION
Work with multidimensional data from your OLAP cubes, data warehouses and Microsoft® SQL Server® Analysis Services.



ASP.NET GAUGE

Whether it's for a sidebar gadget or an internal portal such as SharePoint®, gauges play a crucial role on any dashboard.

Task Name	Duration	Start	Finish
- Root Task 1	8 days	2/9/2011	2/16/2011
- Child Task 1	8 days	2/9/2011	2/16/2011
Sub Task 1	3 days	2/9/2011	2/11/2011
Sub Task 2	2 days	2/9/2011	2/10/2011
Child Task 1	1 day	2/10/2011	2/10/2011
MileStone 1	5 days	2/9/2011	2/13/2011
- Root Task 2	7 days	2/9/2011	2/16/2011
Child Task 2	7 days	2/9/2011	2/15/2011
Child Task 4	5 days	2/9/2011	2/13/2011

Feb 06, '11
W T F S S M

Task
Task: Child Task 2
Start: Thu 2/10/2011
Duration: 1 day
Finish: Thu 2/10/2011

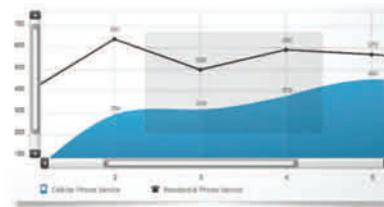
WINDOWS FORMS GANTT CHARTS

Deliver a Microsoft Project-style user experience to your users, with the composite tabular/timeline view of your scheduling data.

Product ID	Product Name	Product Number
Equals	Contains	Contains
Clear Filter	Adjustable Race	AR-5381
Equals	Bearing Ball	BA-8327
Does not equal	BB Ball Bearing	BE-2349
Greater than	Headset Ball Bearings	BE-2908
Less than	Blade	BL-2036
Greater than or equal to	LL Crankarm	CA-5965
Less than or equal to	ML Crankarm	CA-6738
318	HL Crankarm	CA-7457
319	Chainring Bolts	CB-2903
320	Chainring Nut	CN-6137

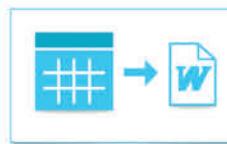
jQUERY

The most robust and forward-thinking product we have based on emerging Web technologies including HTML5 and CSS 3.



CHARTING

Make business charting quick and easy with fast, interactive, and vivid visuals across every .NET platform.



EXPORT TO MICROSOFT® WORD

New class library can create Word documents and stream xamGrid™ contents from Silverlight to a Word document.



SILVERLIGHT DATA VISUALIZATION

Use the Silverlight Data Visualization treemap control to communicate differences in data points with different pattern identifications.



WINDOWS PHONE 7

Visually and functionally designed to build eye-catching, high-end user experiences that take your mobile applications to the next level on the Microsoft® Windows Phone® 7.



SCAN HERE
for an exclusive
look at Ultimate!
www.infragistics.com/ult

TAKE YOUR APPLICATIONS TO THE
NEXT LEVEL WITH OUR TOOLS
INFRAGISTICS.COM/ULTIMATE

 **INFRAGISTICS™**
DESIGN / DEVELOP / EXPERIENCE



dtSearch®

Instantly Search Terabytes of Text

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

InfoWorld

"Covers all data sources ... powerful Web-based engines"

eWEEK

"Lightning fast ... performance was unmatched by any other product"

Redmond Magazine

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

Highlights hits in a wide range of data, using dtSearch's own file parsers and converters

- Supports MS Office through 2010 (Word, Excel, PowerPoint, Access), OpenOffice, ZIP, HTML, XML/XSL, PDF and more
- Supports Exchange, Outlook, Thunderbird and other popular email types, including nested and ZIP attachments
- Spider supports static and dynamic web data like ASP.NET, MS SharePoint, CMS, PHP, etc.
- API for SQL-type data, including BLOB data

25+ full-text & fielded data search options

- Federated searching
- Special forensics search options
- Advanced data classification objects

APIs for C++, Java and .NET through 4.x

- Native 64-bit and 32-bit Win / Linux APIs; .NET Spider API
- Content extraction only licenses available

Desktop with Spider

Network with Spider

Publish (portable media)

Web with Spider

Engine for Win & .NET

Engine for Linux

Ask about fully-functional evaluations!

The Smart Choice for Text Retrieval® since 1991

www.dtSearch.com • 1-800-IT-FINDS



msdn magazine

AUGUST 2011 VOLUME 26 NUMBER 8

LUCINDA ROWLEY Director

KIT GEORGE Editorial Director/mmeditor@microsoft.com

KERI GRASSL Site Manager

KEITH WARD Editor in Chief/mmeditor@microsoft.com

DAVID RAMEL Technical Editor

SHARON TERDEMAN Features Editor

WENDY GONCHAR Managing Editor

KATRINA CARRASCO Associate Managing Editor

SCOTT SHULTZ Creative Director

JOSHUA GOULD Art Director

CONTRIBUTING EDITORS Dino Esposito, Joseph Fultz, Kenny Kerr, Julie Lerman, Dr. James McCaffrey, Ted Neward, Charles Petzold, David S. Platt

RedmondMediaGroup

Henry Allain President, Redmond Media Group

Matt Morollo Vice President, Publishing

Doug Barney Vice President, Editorial Director

Michele Imgrund Director, Marketing

Tracy Cook Online Marketing Director

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

Matt Morollo VP Publishing

Chris Kourtoglou Regional Sales Manager

William Smith National Accounts Director

Danna Vedder Microsoft Account Manager

Jenny Hernandez-Asandas Director Print Production

Serena Barnes Production Coordinator/msdnadproduction@1105media.com

1105 MEDIA

Neal Vitale President & Chief Executive Officer

Richard Vitale Senior Vice President & Chief Financial Officer

Michael J. Valenti Executive Vice President

Abraham M. Langer Senior Vice President, Audience Development & Digital Media

Christopher M. Coates Vice President, Finance & Administration

Erik A. Lindgren Vice President, Information Technology & Application Development

Carmel McDonagh Vice President, Attendee Marketing

David F. Myers Vice President, Event Operations

Jeffrey S. Klein Chairman of the Board

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in US funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: **MSDN Magazine**, PO. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. **POSTMASTER:** Send address changes to **MSDN Magazine**, PO. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the USA. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o **MSDN Magazine**, 4 Venture, Suite 150, Irvine, CA 92618.

Legal Disclaimer: The information in this magazine has not undergone any formal testing by 1105 Media, Inc., and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

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

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

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

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

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

Microsoft



Printed in the USA

techxtend.com
866-719-1528



programmer's paradise



LEADTOOLS Document Imaging Suite SDK v17.0 by LEAD Technologies

- Libraries for C/C++, .NET, Silverlight, Windows Phone, WPF, WCF & WF
- High Performance OCR, ICR, MICR & OMR
- 1D & 2D Barcodes (Read/Write)
- Forms Recognition/Processing
- PDF, PDF/A and XPS
- Document Cleanup
- Advanced Compression (CCITT G3/G4, JBIG2, MRC, ABIC, ABC)
- High-Speed Scanning
- Print Capture and Document Writers

techxtend.com/lead

Paradise #
LOS 03301A02

\$4,018.99



Embarcadero Delphi XE *The Fastest Way to Build Native Windows Applications*

by Embarcadero

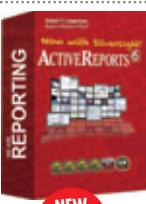
Embarcadero® Delphi® XE is the fastest way to deliver ultra-rich, ultra-fast Windows applications. Dramatically reduce coding time and create applications 5x faster with component-based development and a fully visual two-way RAD IDE. Speed development across multiple Windows and database platforms — for GUI desktop applications, interactive touch-screen, kiosk, and database-driven multi-tier, cloud, and Web applications.

SPECIAL
PRICE!

Paradise #
CGI 32401A01

\$1,977.99

techxtend.com/embacadero



ActiveReports 6 by GrapeCity PowerTools

The de facto standard reporting tool for Microsoft Visual Studio .NET

- Fast and Flexible reporting engine
- Flexible event-driven API to completely control the rendering of reports
- Wide range of Export and Preview formats including Windows Forms Viewer, Web Viewer, Adobe Flash and PDF
- XCopy deployment
- Royalty-Free Licensing for Web and Windows applications

techxtend.com/grapecity

Professional Ed.
Paradise #
D03 04301A01

\$1,310.99



StorageCraft ShadowProtect Desktop Edition 4.0 by StorageCraft Technology Corp

ShadowProtect Desktop provides fast and reliable disaster recovery, data protection and system migration to get desktops and laptops online as quickly as possible. ShadowProtect Desktop is an automated backup that works in the background. In the event of a disaster, its flexible recovery options allow you to complete granular recovery of files and folders or full bare metal recovery in a matter of minutes. Hardware Independent Restore (HIR) technology makes it quick and easy to recover to the same system, to dissimilar hardware or to and from virtual environments.

Minimum
Quantity: 1
Paradise #
SC5 02201E01

\$83.99

techxtend.com/storagecraft

Acronis® Backup & Recovery™ 11 Advanced Server by Acronis

**NEW
VERSION
11!**

Acronis Backup & Recovery 11 Advanced Server provides disk-based backup, disaster recovery and data protection — with support for disk, tape and cloud storage options. It is a unified solution that may be used either separately or in combination with other products: all managed from the same console, with the same backup plans, reports and storage! It includes both "Agent for Windows" and "Agent for Linux," which simplifies the purchasing process by giving you flexibility of using either.



Upgrade
1-9 Users
Paradise #
AT2 43501SO1

\$785.99

techxtend.com/acronis

Apple MacBook Pro MC721LL/A 15.4" LED Notebook by Apple

MacBook Pro is machined from a single piece of aluminum, an engineering breakthrough that replaced many parts with just one. It's called the unibody. And the first time you pick up a MacBook Pro you'll notice the difference it makes. The entire enclosure is thinner and lighter than other notebooks. It looks polished and refined. And it feels strong and durable — perfect for life inside (and outside) your briefcase or backpack.



Paradise #
ZHI B10001

\$1,799.00

techxtend.com/apple

HP Color LaserJet CP3525dn Printer by Hewlett Packard

Easily produce eye-catching black and color documents with this high-performing, reliable printer for general and specialty print projects!

You can get black prints for the same cost as on a black-and-white HP LaserJet — providing general office effectiveness, while also allowing you to print in color! Choose an optional high-capacity print cartridge for even more value. With HP's Color Access Control technology, easily control who can print in color and how much.



Paradise #
ZHI U09885

\$894.99

techxtend.com/hp

VMware vSphere Essentials Kit Bundle

vSphere Essentials provides an all-in-one solution for small offices to virtualize three physical servers for consolidating and managing applications to reduce hardware and operating costs with a low up-front investment. vSphere Essentials includes:

- VMware ESXi and VMware ESX (deployment-time choice)
- VMware vStorage VMFS
- Four-way virtual SMP
- VMware vCenter Server Agent
- VMware vStorage APIs/VCB
- VMware vCenter Update Manager
- VMware vCenter Server for Essentials



for 3 hosts
Paradise #
V55 85101C02

\$446.99

techxtend.com/vsphere

TX Text Control 16.0

Word Processing Components

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

- .NET WinForms and WPF rich text box 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 12101A01

\$1,109.99

[Download a demo today.](http://techxtend.com/textcontrol)
techxtend.com/textcontrol



Lenovo ThinkPad X220 4290

by Lenovo

Engineered for mobile professionals, the ThinkPad X220 features the quality and performance Lenovos users have come to expect and increased audio, video and communications features that respond to the increased use of laptops as a multimedia and communications tools in business. The ultra-portable ThinkPad X220 comes equipped with a full-powered Intel processor with outstanding graphic performance.

Paradise #
ZHI GF0801

\$2,169.00

techxtend.com/lenovo

Programmer's Paradise is getting a new name: TechXtend!



For nearly 30 years, Programmer's Paradise has served the software development and IT communities with the best selection of software, terrific values and a commitment to service excellence.

30 years... much has changed in that time!

However, one thing that won't ever change is that we will still be your "go to" source for software developer tools — AND you'll also be able to depend on TechXtend for all of your other IT needs.

Learn more about our new name: www.techxtend.com/techxtend



Win an iPad!

NO PURCHASE NECESSARY. Use offer code WEBTRW08 when you place your order online or with your TechXtend/Programmer's Paradise representative and you'll automatically be entered into a drawing to win an iPad Wi-Fi 32GB.

For official rules, or to complete the online entry form: www.techxtend.com/tradewinds

Software, Systems, & Solutions – Since 1982

Prices subject to change. Not responsible for typographical errors.



EDITOR'S NOTE

KEITH WARD

F-Sharp Focus

F# is a growing and increasingly important programming language. Many developers are using it every day; many more should be using it on a regular basis.

Being busy types, though, it can be difficult to find time to learn a new language—it's hard enough getting the daily work done. That's why I've asked Don Syme, the creator of F#, and his crack team to answer some common F# questions I hear folks asking.

Why was F# created? What need does it serve?

The modern enterprise has a range of software needs that don't fit the traditional "business apps" IT model. For example, financial modeling involves quantitative finance domain experts ("quants") who need to write and deploy models that analyze the value and risk of market positions. You can think of these people as one example of a domain-oriented programmer.

There are many similar examples in modern enterprises and start-ups, including machine-learning, statistical, parallel, scientific and algorithmic experts; these people are often at the core of modern software teams.

Domain programmers are always on a search for tools that improve productivity, performance and robustness in balance and often utilize analytical scripting languages as part of their work. Dynamically typed scripting languages are great in many ways, but can lead to real problems. For example, they can lead to problems as applications grow in size. Often, they don't have good visual tooling—for instance, no IntelliSense—and may not be very performant when fresh code is written in the language. Finally, components don't live in isolation, and ultimately need to be deployed as encapsulated software components in the context of larger .NET systems.

When you look at examples across the industry, the common need is for a simple, succinct, efficient and typed language that allows math-oriented and domain experts to work in their problem domain—rather than get lost in a sea of object-oriented (OO) class-oriented code—while still contributing professional software components that interoperate well. That's where F# fits in. Ultimately, this kind of tool makes domain experts happier and more productive members of software teams.

The special thing about F# is that we manage to get most of the benefits of dynamic language (rapid development and succinctness)

while also still retaining the benefits of strongly typed languages (robustness, performance and visual tooling), all in the context of a highly interoperable platform.

What are the most common scenarios for use of F#?

F# can be used for just about anything, but it excels at what we call "analytical component development." This includes financial- and market-analysis engines (including event-driven ones), server-side machine learning components and other data-rich analyses. There are several good case study examples available on the Microsoft Web sites.

Some interesting examples outside finance are the uses of F# in energy trading, Bing advertising, Project Emporia (a personalized news-selection service based on machine learning techniques) and in the artificial intelligence engine of the "Path of Go" game. All of these show where F# excels: analytical engines in the context of professional .NET software delivery.

What are the main advantages

F# has over C# and C++?

F# differs in many important ways from C#. Technically speaking, some of the things you'll notice are the functional-first methodology and language features (such as tuples, lists and pattern-matching); the strong focus on immutable data; the inclusion of key typing features relevant to scientific and numerical programming (including F# units of measure); the increased use of expressions as a form of software re-use (including language features such as object expressions); the succinct representation of OO programming; the declarative and compositional model of asynchronous programming (a version of which is appearing in the next version of C#); and the inclusion of a dynamic interactive compiler called F# Interactive for scripting.

F# can be surprisingly intuitive for C and C++ programmers. Perhaps the thing they'll notice most is the immediacy of programming with F# Interactive—some C++ people are surprised that it's so easy to get your hands on data and play with it in a strongly typed language, while still getting great performance. Software becomes fun and explorative again.

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

© 2011 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, *MSDN*, and Microsoft logos are used by 1105 Media, Inc. under license from owner.

NEW!



OnTime11 is Here!

Ship Software OnTime

Project Management • Bug Tracking • Agile/Scrum

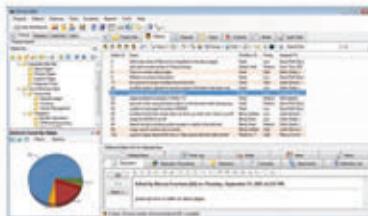
Get 2 Users Free... Forever! • Visit www.axosoft.com

Hosted



OnTime Now! Managing an Agile or Scrum dev team using a cloud-based solution has never been easier or more cost effective. Get started with 2 free users in just seconds at axosoft.com!

Windows



OnTime for Windows is installed in your own server environment. It uses a .NET & SQL back-end, integrates easily with Visual Studio, and is a total joy to use in the rich Windows client. Includes free SDK / APIs.

Web



OnTime Web provides you with an anywhere-anytime solution, whether you go with OnTime hosted or installed. This is what a web app is supposed to be - an intuitive UI, blazing fast reaction, and a powerful back end.

Free 2-user License • Free Download • Free Web Demo



Static Code Analysis and Code Contracts

For many, it's hard to believe that there was ever a time when you could print the entire source code of an application, read it from top to bottom and catch bugs or possible misbehavior. The complexity of modern software makes this approach impractical for humans—but not for some types of smart software, such as static-analysis tools.

Static analysis of software consists of inferring the behavior of a piece of code from its instructions. A static-analysis tool conducts its examination of source code by progressively building a base of knowledge that it uses to prove statements and their output right or wrong. Static analysis doesn't actually execute the code; it just reads lines and figures things out. What kind of things?

In general, static analysis can identify possible bugs in code as well as indicate how closely certain segments of code match expectations and specifications.

It has been known for some time—at least as far back as the great work done by Alan Turing in the 1930s—that there's no automatic and completely reliable way to decide whether a given program will terminate without errors. In any case, computer science is mostly based on approximation, and being forewarned that you likely have a bug around some lines is a valuable help. Scale this help to the size of a modern project and you can see the real value. A static-analysis tool probably won't give you absolute certainties but, working in real time as you write your classes, it can give you immediate feedback about what may be wrong in your implementation.

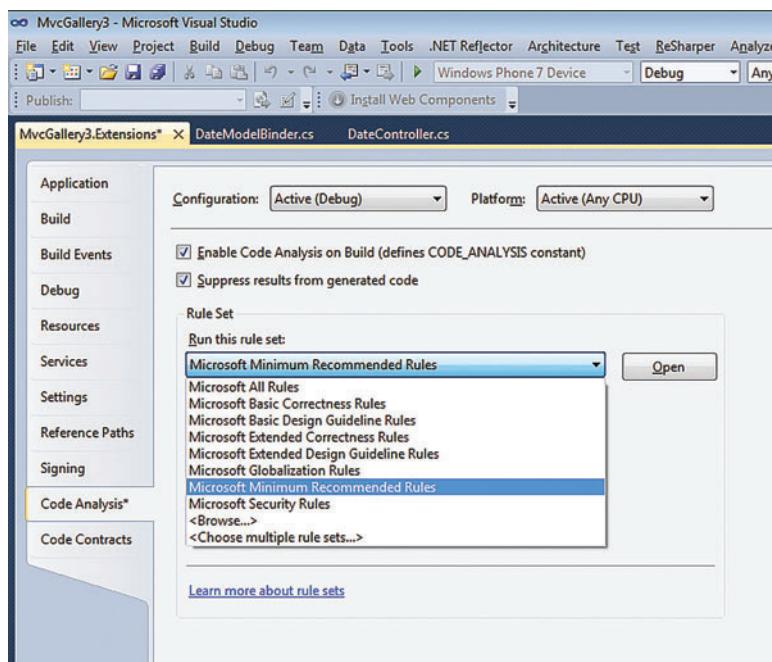


Figure 1 Code Analysis Settings in a Visual Studio 2010 Project

Static Analysis and Testing

There's a fundamental difference between testing and static analysis. As a developer, you actively write the test but passively endure the static analysis. If your tests don't actually cover significant conditions, or don't check them with significant values, the results won't be meaningful. A tool for static analysis gives you warnings about facts (as the tool understands them) that violate some of the configured internal rules. In general, getting nearly no warnings from static analysis is a good indication of the quality of the software. On the other hand, getting warnings doesn't automatically mean your software is buggy and will fail at first run. Static analysis will likely detect hard-to-catch, corner-case errors that have the potential to crash your application. As with testing, static analysis can catch defects quite early in the development phase, thus limiting the impact of software errors on the overall project.

Types of Static Analyzers

Static analysis has many facets and a variety of different levels of implementation. A very basic type of static analyzer is the language compiler. The compiler goes through your source code and matches it to the syntax rules of the language. This is the first step of analysis. Modern compilers, however, do a lot more.

The latest C# compiler, for example, detects violations of the Liskov principle when you sum up Code Contracts in derived classes. This compiler can often point out occurrences of unused variables. However, what a compiler detects and classifies as a warning doesn't necessarily qualify as a bug. Still, the compiler is a distinct step you have to execute. And it may take a while, depending on the project and the environment.

Tools like FxCop or, better yet, the Code Analysis feature in Visual Studio 2010, perform a more specific analysis of the code and can be configured to run either on demand or in each build, as Figure 1 shows.

Moreover, you can declaratively set the rules you want it to employ when processing your code. Figure 2 shows a sample of the long list of warnings you may receive when you enable the All Rules option in the Visual Studio 2010 Code Analysis. Note that some of those warnings are very specific, and very useful for making your code cleaner and more adherent to the Microsoft .NET Framework guidelines.

The key point about static analysis is that automatic tools are absolutely honest and give you warnings based strictly on violations of rules. Detected rule violations are not necessarily bugs; but if a violation is a bug, it likely will show in edge cases, and in full accordance with Murphy's Law—just when it'll do the most damage.

LEADTOOLS® 17.5

LEADTOOLS provides developers easy access to decades of expertise in color, grayscale, document, medical, vector and multimedia imaging.

OCR/OMR

BARCODE

PDF & PDF/A

MEDICAL 3D

DICOM

PACS

MPEG-2 TRANSPORT STREAM



ANNOTATIONS

VIEWER CONTROLS

FORMS RECOGNITION & PROCESSING

FILE FORMATS

VIRTUAL PRINTER

DOCUMENT CLEANUP & PREPROCESSING

SCANNING

DVD & DVR

DIRECTSHOW CODECS

MULTIMEDIA PLAYBACK & CAPTURE

IMAGE PROCESSING

MEDICAL WORKSTATION FRAMEWORK

MEDICAL WEB VIEWER FRAMEWORK

MAJOR ADDITIONS INCLUDE

PDF READER AND VIEWER WITH TEXT EXTRACTION, BOOKMARKS AND ANNOTATIONS. A FASTER OCR ENGINE WITH MORE ACCURACY AND LANGUAGES. A NEW HIGH LEVEL BARCODE INTERFACE WITH SUPPORT FOR .NET, SILVERLIGHT AND WINDOWS PHONE. A NEW FRAMEWORK FOR CREATING CLOUD APPLICATIONS.

.NET

C/C++

SILVERLIGHT/WINDOWS PHONE

ASP.NET

WPF

WF

WCF

FREE 60 DAY EVALUATION - TRY IT TODAY!!!

800 637-1840

WWW.LEADTOOLS.COM

LEAD Technologies' newly released LEADTOOLS Version 17.5 is packed with new features and enhancements delivering more speed, power and extensibility into the hands of application developers than ever before.

Description	File	Line	Column	Project
CA1025 : Microsoft.Design : Change the type or parameter 'baseUri' or method 'PagerHelper.Pager' from string to System.Uri, or provide an overload of 'PagerHelper.Pager(this HtmlHelper, string, int, int, string, int, object)', that allows 'baseUrl' to be passed as a System.Uri object.	Pager.cs			MvcGallery3.Extensions
CA1026 : Microsoft.Design : Replace method 'SimpleGridHelper.SimpleGrid<T>(this HtmlHelper, ICollection<T>, IList<IDataTransformer>)' with an overload that supplies all default arguments.	Grid.cs	12		MvcGallery3.Extensions
CA1801 : Microsoft.Usage : Parameter 'helper' of 'SimpleGridHelper.SimpleGrid<T>(this HtmlHelper, ICollection<T>, IList<IDataTransformer>)' is never used. Remove the parameter or use it in the method body.	Grid.cs	12		MvcGallery3.Extensions
CA1802 : Microsoft.Usage : Parameter 'transformers' of 'SimpleGridHelper.SimpleGrid<T>(this HtmlHelper, ICollection<T>, IList<IDataTransformer>)' is never used. Remove the parameter or use it in the method body.	Grid.cs	12		MvcGallery3.Extensions
CA1062 : Microsoft.Design : In externally visible method 'SimpleGridHelper.SimpleGrid<T>(this HtmlHelper, ICollection<T>, IList<IDataTransformer>)'; validate parameter 'dataSource' before using it.	Grid.cs	25		MvcGallery3.Extensions
CA1803 : Microsoft.Usage : Parameter 'helper' of 'SubmitButtonHelper.Submit(this HtmlHelper, string, string, object)' is never used. Remove the parameter or use it in the method body.	Submit.cs	22		MvcGallery3.Extensions
CA1019 : Microsoft.Design : Add a public read-only property accessor for positional argument 'transformerType' of Attribute 'TransformerAttribute'.	TransformerAttribute.cs	6		MvcGallery3.Extensions

Figure 2 Warnings Received When All Rules Are Enabled

Besides on-demand tools like compilers and FxCop-like facilities, another category of tools can provide a lot of feedback for you—but they work asynchronously. To this category belongs ReSharper, which provides the same type of handy suggestions as Code Analysis, but dynamically as you type. It also offers to edit the code for you at the cost of a click or two. ReSharper detects, for example, unassigned variables, unreachable code, possible null references and a variety of code smells such as virtual calls in a constructor and access to modified closures in LINQ expressions. More importantly, ReSharper allows you to formalize your own code smells and automatically find and replace them intelligently as you type them. For more information and a detailed step-by-step example of the ReSharper structural search and replace, read the blog post at bit.ly/eCMvCL.

Finally, static analysis in the .NET Framework 4 and Visual Studio 2010 also takes place using the Static Checker tool created by the Code Contracts team.

Static Checker in Action

Along with Code Contracts, Microsoft provides the Static Checker tool, which can go through your code even without you compiling it and highlight Code Contracts not being verified or just impossible to prove right or wrong. You can get the Static Checker only if you have Visual Studio 2010 Premium or Ultimate or Visual Studio Team System 2008. These version requirements are in line with the requirements for the Code Analysis feature—that is, none of these features are available in Visual Studio Professional or Express editions.

Like the Intermediate Language rewriter, the Static Checker is a separate download that enhances Visual Studio 2010. You can get the latest Code Contracts bits from bit.ly/fF3wzL. You'll need to explicitly enable Static Checking on a per-project configuration basis, as shown in Figure 3.

The Static Checker can run in the background if you like, and can explicitly show squiggles for each warning. Background execution means that the Static Checker is scheduled to run in parallel with the compiler as a build happens and code is changed.

You can have the Static Checker look for implicit as well as explicit contracts. An explicit contract is any contract you declare in a class method, such as a precondition, a postcondition or an invariant. In addition, the Static Checker can also take into

account some built-in Code Contracts and check whether array indexes always stay within array bounds, whether any null references are used or if any division by zero will occur. You control implicit Code Contracts through the checkboxes shown in Figure 3. Note that enabling implicit Code Contracts may flood your output window with a ton of messages and warnings. You might not want to do this all the time, and probably not after the beginning.

Here's an example of code that will be caught as a potential error when you have the Implicit Array Bounds Obligations contract on:

```
var numbers = new int[2];
numbers[3] = 0;
```

Most of the power of the Static Checker, though, results from using it with explicit contracts, such as preconditions and postconditions.

Explicit Contracts

Suppose you have the following code in one of your methods:

```
var orderId = GetLatestOrderId();
ProcessOrder(orderId);
```

A variable is obtained from a function call and then passed on to another method for further processing. There's not much any Static Checker can do to try to prove these statements right or wrong without some explicit contract information. On the other hand, if the GetLatestOrderId method exposes a meaningful postcondition, as in the following code, the Checker might be able to prove whether the successive call is valid:

```
public int GetLatestOrderId()
{
    Contract.Ensures(Contract.Result<int>() > 0);
    ...
    return n;
}
```

The GetLatestOrderId method explicitly declares it will be returning a value greater than zero. As the Checker makes its pass, it grabs this piece of information and adds it to its knowledgebase. Later on, when it finds that the return value from GetLatestOrderId method

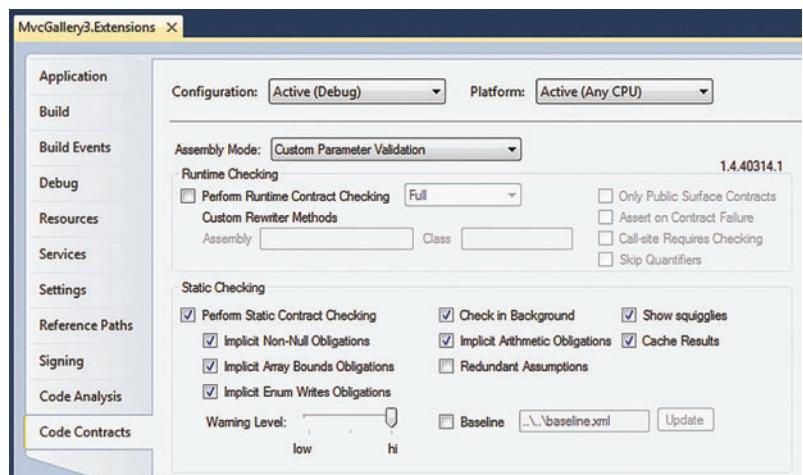


Figure 3 Enabling Static Checking of Code Contracts in Visual Studio 2010



NetAdvantage® for jQuery

A screenshot of the IG GRID component on an HTML page. It displays a table of products with columns for Product ID, Product Name, Product Number, and Standard Cost. The grid includes filtering and sorting controls at the top. A sidebar on the left shows filter criteria for Product Name, such as 'Contains' and 'Equals'. The data includes items like Adjustable Race, Bearing Ball, and Headset Ball Bearings.

IG GRID ON HTML PAGE

High performance grid lets users page through any OData source, even streaming movie descriptions.

A screenshot of the IG GRID component displaying movie reviews from Netflix's OData service. The grid shows columns for Movie Name, Image, and Movie Synopsis. It includes a sidebar for filtering and sorting. Below the grid, there are movie reviews with small thumbnail images and descriptions, such as "U.S. Marshals" and "The Mask of Zorro".

IG EDITORS

Robust data entry and editing controls assist users with entering data in the formats your application needs.

A screenshot of the IG EDITORS component showing a form for entering product price information. It includes fields for First Name, Last Name, Credit Card Number, Expiration Date, Country, Price, Quantity, Discount, and Total. A 'Submit' button is at the bottom.

SCAN HERE
for an exclusive
look at jQuery!
www.infragistics.com/jq

IG GRID ALL FEATURES ENABLED

The grid's sophisticated filtering and sorting features allow your users to find the records they need to work with.

A screenshot of the IG RATING component showing a list of movies with star rating controls. The grid includes columns for Movie and Rating. Each movie entry has a set of five stars that can be rated by the user. Examples include "The Shawshank Redemption (1994)", "Star Wars: Episode V (1980)", and "Raiders of the Lost Ark (1981)".

RATING

Enable a social experience for users, by letting them rate their favorite movies or anything else with our rating control.

A screenshot of the IG VIDEO PLAYER MVC component showing a video player interface. It displays a video thumbnail for "Big Buck Bunny" and a progress bar. The video title "Big Buck Bunny" is prominently displayed in the center.

IG VIDEO PLAYER MVC

When a user finds what they want to watch, our HTML5 video player adds streaming video right into your own apps.



UPLOADERS

Accept file uploads of any kind of content straight from your users with background file transfers.

TAKE YOUR WEB APPLICATIONS TO THE NEXT LEVEL WITH OUR TOOLS
INFRAGISTICS.COM/JQUERY

INFRASTICS™
DESIGN / DEVELOP / EXPERIENCE

Error List				
0 Errors	2 Warnings	1 Message		
Description	File	Line	Column	Project
⚠ 1 CodeContracts: requires unproven: orderId > 0	Program.cs	27	13	LiskovWarning
⚠ 2 + location related to previous warning	Program.cs	32	13	LiskovWarning
ⓘ 3 CodeContracts: Checked1 assertion: 1 unknown	LiskovWarning.exe	1	1	LiskovWarning

Figure 4 Unproven Contracts

is being used as input to another method with an explicit precondition, the Checker can reasonably attempt to see if the operation is consistent with declared contracts. Let's consider the following precondition contract:

```
public void ProcessOrder(int orderId)
{
    Contract.Requires<ArgumentException>(orderId > 0);
    ...
}
```

These two explicit contracts provide enough information to the Checker to prove the contract. In this case, you won't be getting any extra warning. But what if some of these contracts are missing? The Static Checker returns an output similar to what you see in **Figure 4**. The Checker detects that, say, a method has a Requires contract and receives data from another one that doesn't say anything about its returned output.

It should be clear by now that, to get the most out of the Static Checker, you must use Code Contracts all throughout your code. The more explicit contracts you bring in, the more reliable the Static Checker can be.

Assume and Assert

Sometimes, however, you have to integrate your new code with legacy code that others have written and that can't be updated. If the legacy code doesn't have any contracts, you're going to get annoying warnings from the Static Checker. The Code Contracts API offers a workaround for this.

Essentially, unproven contract warnings originate from lack of information—the Checker is unable to find the required information. However, as the developer, you can provide more details—specifically, assumptions. Have a look at **Figure 5**.

You see a tooltip that warns about a possible null reference. Apparently, the variable *context* is being used without having been assigned. Who's actually responsible for the tooltip? Is it the Checker or is it something else?

```
// Get the value from any of the input collections
Contract.Assume(context != null);
var key = String.Format("{0}.{1}", context.ModelName, id);
Possible 'System.NullReferenceException'

var result = context.ValueProvider.GetValue(key);
if (result == null && context.FallbackToEmptyPrefix)
{
    // Try without prefix
    result = context.ValueProvider.GetValue(id);
    if (result == null)
        return default(T);
}
```

Figure 5 A Warning from the Static Checker

In this case, the tooltip comes from ReSharper, whose analysis engine correctly determines a possible null reference. Why don't we get an analogous message from the Checker? That's because of the first line in **Figure 5**:

```
Contract.Assume(context != null);
```

With this instruction, you guarantee to the Checker that the variable

context is never null. The Checker just trusts you and adds that piece of information to its knowledgebase. ReSharper doesn't currently provide full support for .NET Code Contracts, which explains why it still gives you a warning. On the other hand, ReSharper does support its own set of annotations that you can use in much the same way as with assumptions. See bit.ly/VSnkj for more details.

Optimal Use of Static Analysis

Static analysis is difficult, and it's probably not even an exact science. It often happens that you fill your code with contract information with the best intentions, just to find out that it significantly slows the build process. There are a couple of approaches to consider to optimize the use of contracts and subsequent analysis.

First, you might want to create a special build configuration and enable static checking only on that. You switch to it periodically, grab the feedback and apply it. When you're done, you move back to build your solution as usual without the additional burden of static analysis.

Second, you can try using contracts piecemeal. You apply contracts extensively in the code, but then you disable contracts at the assembly level. You can do this by simply adding the following attribute to the properties of the assembly:

```
[assembly: ContractVerification(false)]
```

Next, you re-enable contract checking only where you're currently focusing: class, method or assembly. You use the same attribute with a value of true.

Wrapping Up

Static analysis is a technique that aims to evaluate the correctness of your source code without running it. There are a few tools that do this to various extents. One is simply the compiler; another analysis tool is the Static Checker—an executable that usually integrates with the build process. In the .NET Framework, a special Static Checker tool works by learning facts about your code from Code Contracts. The Checker then evaluates these facts and highlights possible errors and contract violations.

At a time when complexity increases continually and development teams always run short of time, integrated static analysis saves a bit of time in builds and, more importantly, saves you from nasty bugs that hit your software just in corner cases. ■

DINO ESPOSITO is the author of “Programming Microsoft ASP.NET MVC” (Microsoft Press, 2010) and coauthor of “Microsoft .NET: Architecting Applications for the Enterprise” (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. Follow him on Twitter at twitter.com/despos.

THANKS to the following technical expert for reviewing this article:
Manuel Fahndrich



NetAdvantage[®]

for Data Visualization



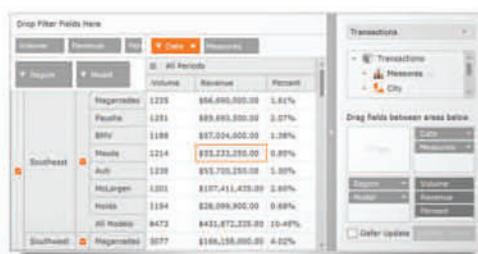
INTERACTIVE DASHBOARDS

Build rich dashboards that visualize business data to empower decision makers.



MEDIA TIMELINE

The timeline highlights how easily users can search, browse and play popular videos from YouTube.



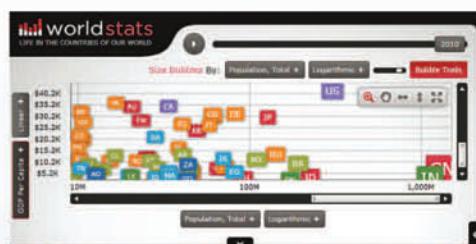
OLAP PIVOT GRID DATA VISUALIZATION

Work with multidimensional data from your OLAP cubes, data warehouses and Microsoft SQL Server[®] Analysis Services.



MAP OUT ANYTHING AND EVERYTHING

Use xamMap™ to get an instant view and show anything including seating charts, floor plans, warehouse contents, and yes—geographic maps, too!



INFRASTRIKS MOTION FRAMEWORK™

Create an immersive and animated user experience that tells the story of your data over time like no other visualization can.

TAKE YOUR APPLICATIONS TO
THE NEXT LEVEL WITH OUR TOOLS
INFRAGISTICS.COM/DV



Infragistics Sales 800 231 8588 • Infragistics Europe Sales +44 (0) 800 298 9055 • Infragistics India +91 80 4151 8042 • @infragistics

Copyright 1996-2011 Infragistics, Inc. All rights reserved. Infragistics and NetAdvantage are registered trademarks of Infragistics, Inc. Motion Framework and xamMap are trademarks of Infragistics, Inc. The Infragistics logo is a trademark of Infragistics Inc.

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE



The Windows Thread Pool and Work

Concurrency means different things to different people. Some folks think in terms of agents and messages—cooperating but asynchronous state machines. Others think in terms of tasks, usually in the form of functions or expressions that may execute concurrently. Still others think in terms of data parallelism, where the structure of the data enables concurrency. You might even consider these complementary or overlapping techniques. Regardless of how you view the world of concurrency, at the heart of any contemporary approach to concurrency is a thread pool of one form or another.

Threads are relatively expensive to create. An excessive number of threads introduces scheduling overhead that affects cache locality and overall performance. In most well-designed systems, the unit of concurrency is relatively short-lived. Ideally, there's a simple way to create threads as needed, reuse them for additional work and avoid creating too many threads in some intelligent way in order to use the available computing power efficiently. Fortunately, that ideal exists today, and not in some third-party library, but right in the heart of the Windows API. Not only does the Windows thread pool API meet these requirements, but it also integrates seamlessly with many of the core building blocks of the Windows API. It takes much of the complexity out of writing scalable and responsive applications. If you're a longtime Windows developer, you're undoubtedly familiar with the cornerstone of Windows scalability that's the I/O completion port. Take comfort in the fact that an I/O completion port sits at the heart of the Windows thread pool.

At the heart of any contemporary approach to concurrency is a thread pool of one form or another.

Keep in mind that a thread pool shouldn't be viewed simply as a way to avoid calling `CreateThread` with all of its parameters and the requisite call to `CloseHandle` on the resulting handle. Sure, this may be convenient, but it can also be misleading. Most developers have expectations about the priority-driven, preemptive scheduling model that Windows implements. Threads at the same priority will typically share processor time. When a thread's quantum—the

amount of time it gets to run—comes to an end, Windows determines whether another thread with the same priority is ready to execute. Naturally, many factors influence thread scheduling, but given two threads that are created around the same time, with the same priority, both performing some compute-bound operation, one would expect them both to begin executing within a few quanta of each other.

Not so with the thread pool. The thread pool—and really any scheduling abstraction based on an I/O completion port—relies on a work-queuing model. The thread pool guarantees full core utilization but also prevents overscheduling. If two units of work are submitted around the same time on a single-core machine, then only the first is dispatched. The second will only start if the first finishes or blocks. This model is optimal for throughput because work will execute more efficiently with fewer interruptions, but it also means that there are no latency guarantees.

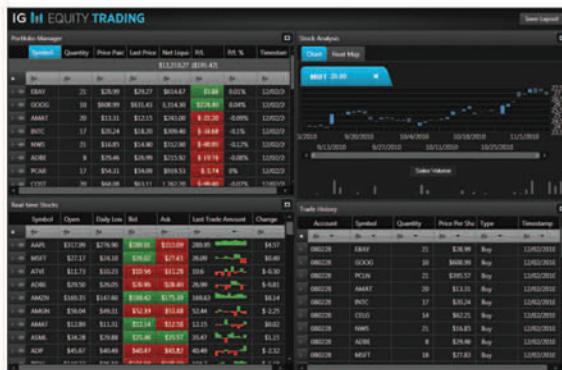
The thread pool API is designed as a set of cooperating objects. There are objects representing units of work, timers, asynchronous I/O and more. There are even objects representing challenging concepts such as cancellation and cleanup. Fortunately, the API doesn't force developers to deal with all of these objects and, much like a buffet, you can consume as little or as much as needed. Naturally, this freedom introduces the risk of using the API inefficiently or in an inappropriate way. That's why I'll be spending the next few months on it in this column. As you begin to grasp the different roles that the various parts of the API play, you'll discover that the code you need to write gets simpler rather than more complex.

In this first installment, I'm going to show you how to start submitting work to the thread pool. Functions are exposed to the thread pool as work objects. A work object consists of a function pointer as well as a void pointer, called a context, which the thread pool passes to the function every time it's executed. A work object can be submitted multiple times for execution, but the function and context can't be changed without creating a new work object.

The `CreateThreadpoolWork` function creates a work object. If the function succeeds, it returns an opaque pointer representing the work object. If it fails, it returns a null pointer value and provides more information via the `GetLastError` function. Given a work object, the `CloseThreadpoolWork` function informs the thread pool that the object may be released. This function doesn't return a value, and for efficiency assumes the work object is valid.

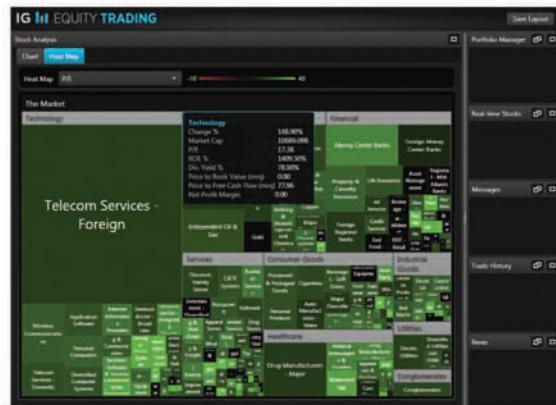
NetAdvantage®

PERFORMANCE



REAL-TIME USER INTERFACES

Refresh your app's user interface tick-by-tick on every value change in the fastest applications imaginable.



ACROSS ALL PLATFORMS

Charts and grids featuring blazing speed whether you are using ASP.NET, Silverlight, WPF or any other .NET platform.



DEEP DRILLDOWN

Dive into the deepest details of your data with crisp, rapidly-updating visual controls designed to handle it all.



SCAN HERE
for an exclusive
look at Performance!
www.infragistics.com/perf

TAKE YOUR APPLICATIONS TO
THE NEXT LEVEL WITH OUR TOOLS
INFRAGISTICS.COM/PERFORMANCE



INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

Fortunately, the unique_handle class template I introduced in last month's column takes care of this. Here's a traits class that can be used with unique_handle, as well as a typedef for convenience:

```
struct work_traits
{
    static PTP_WORK invalid() throw()
    {
        return nullptr;
    }

    static void close(PTP_WORK value) throw()
    {
        CloseThreadpoolWork(value);
    }
};

typedef unique_handle<PTP_WORK, work_traits> work;
```

I can now create a work object and let the compiler take care of its lifetime, whether the object resides on the stack or in a container. Of course, before I can do so, I need a function for it to call, known as a callback. The callback is declared as follows:

```
void CALLBACK hard_work(PTP_CALLBACK_INSTANCE, void * context, PTP_WORK);
```

The CALLBACK macro ensures that the function implements the appropriate calling convention that the Windows API expects for callbacks, depending on the target platform. Creating a work object for this callback using the work typedef is straightforward and continues the pattern I highlighted in last month's column, as shown here:

```
void * context = ...
work w(CreateThreadpoolWork(hard_work, context, nullptr));
check_bool(w);
```

At this point, all I have is an object that represents some work to perform, but the thread pool itself isn't yet involved, as the work callback hasn't been submitted for execution. The SubmitThreadpoolWork function submits the work callback to the thread pool. It may be called multiple times with the same work object to allow multiple callbacks to run concurrently. The function is shown here:

```
SubmitThreadpoolWork(w.get());
```

Of course, even submitting the work doesn't guarantee its prompt execution. The work callback is queued, but the thread pool may limit the level of concurrency—the number of worker threads—to improve efficiency. As this is all rather unpredictable, there needs to be a way to wait for outstanding callbacks, both those that may be currently executing as well as those that are still pending. Ideally, it would also be possible to cancel those work callbacks that have yet to be given an opportunity to execute. Usually any sort of blocking "wait" operation is bad news for concurrency, but it's still necessary in order to perform predictable cancellation and shutdown. That's the topic of an upcoming column, so I won't spend much more time on it here. However, for now, the WaitForThreadpoolWorkCallbacks function meets the aforementioned requirements. Here's an example:

```
bool cancel = ...
WaitForThreadpoolWorkCallbacks(w.get(), cancel);
```

The value of the second parameter determines whether pending callbacks will be canceled or whether the function waits for them to complete even if they haven't yet begun to execute. I now have enough to build a basic functional pool, taking the thread pool API and a sprinkling of C++ 2011 to build something that's a lot more enjoyable to use. Moreover, it provides a good example for using all of the functions I've introduced thus far.

A simple functional pool should allow me to submit a function to execute asynchronously. I should be able to define this function using a lambda expression, a named function or a function object, as needed. One approach is to use a concurrent collection to store a queue of functions, passing this queue to a work callback. Visual C++ 2010 includes the concurrent_queue class template that will do the trick. I'm assuming that you're using the updated implementation from Service Pack 1, as the original had a bug that resulted in an access violation if the queue wasn't empty upon destruction.

I can go ahead and start defining the functional pool class as follows:

```
class functional_pool
{
    typedef concurrent_queue<function<void()>> queue;

    queue m_queue;
    work m_work;

    static void CALLBACK callback(PTP_CALLBACK_INSTANCE, void * context, PTP_WORK)
    {
        auto q = static_cast<queue *>(context);

        function<void()> function;
        q->try_pop(function);

        function();
    }
};
```

As you can see, the functional_pool class manages a queue of function objects as well as a single work object. The callback assumes that the context is a pointer to the queue and further assumes that at least one function is present in the queue. I can now create the work object for this callback and set the context appropriately, as shown here:

```
public:

functional_pool():
    m_work(CreateThreadpoolWork(callback, &m_queue, nullptr))
{
    check_bool(m_work);
}
```

A simple functional pool should allow me to submit a function to execute asynchronously.

A function template is needed to cater to the various types of functions that may be submitted. Its job is simply to queue the function and call SubmitThreadpoolWork to instruct the thread pool to submit the work callback for execution, as shown here:

```
template <typename Function>
void submit(Function const & function)
{
    m_queue.push(function);
    SubmitThreadpoolWork(m_work.get());
}
```

Finally, the functional_pool destructor needs to ensure that no further callbacks will execute before allowing the queue to be destroyed, otherwise horrible things will happen. Here's an example:

```
~functional_pool()
{
    WaitForThreadpoolWorkCallbacks(m_work.get(), true);
}
```

I can now create a functional_pool object and submit work quite simply using a lambda expression:

```
functional_pool pool;  
  
pool.submit([]  
{  
    // Do this asynchronously  
});
```

There's going to be some performance penalty for explicitly queuing functions and implicitly queuing work callbacks.

Clearly, there's going to be some performance penalty for explicitly queuing functions and implicitly queuing work callbacks. Using this approach in server applications, where the concurrency is typically quite structured, would probably not be a good idea. If you have only a handful of unique callbacks that handle the bulk of your asynchronous workloads, you're probably better off just using function pointers. This approach may be useful in client applications, however. If there are many different short-lived operations that you'd like to handle concurrently to improve responsiveness, the convenience of using lambda expressions tends to be more significant.

Anyway, this article isn't about lambda expressions but about submitting work to the thread pool. A seemingly simpler approach for achieving the same end is provided by the TrySubmitThreadpoolCallback function, as shown here:

```
void * context = ...  
check_bool(TrySubmitThreadpoolCallback(  
    simple_work, context, nullptr));
```

It's almost as if the CreateThreadpoolWork and SubmitThreadpoolWork functions have been rolled into one, and that's essentially what's happening. The TrySubmitThreadpoolCallback function causes the thread pool to create a work object internally whose callback is immediately submitted for execution. Because the thread pool owns the work object, you don't have to concern yourself with releasing it. Indeed, you can't, because the work object is never exposed by the API. The callback's signature provides further evidence, as shown here:

```
void CALLBACK simple_work(  
    PTP_CALLBACK_INSTANCE, void * context);
```

The callback looks much the same as before except for the missing third parameter. At first, this seems ideal: a simpler API and less to worry about. However, there's no obvious way to wait for the callback to

complete, let alone to cancel it. Trying to write the functional_pool class in terms of TrySubmitThreadpoolCallback would be problematic and require additional synchronization. An upcoming column addresses how this can be achieved using the thread pool API. Even if you were able to solve these issues, a less obvious problem exists that's potentially far more devastating in practice. Every call to TrySubmitThreadpoolCallback involves the creation of a new work object with its associated resources. With heavy workloads, this can quickly cause the thread pool to consume a great deal of memory and result in further performance penalties.

Using a work object explicitly also provides other benefits. The callback's final parameter in its original form provides a pointer to the same work object that submitted the running instance. You can use it to queue up additional instances of the same callback. You can even use it to release the work object. However, these sorts of tricks can get you into trouble, as it becomes increasingly difficult to know when it's safe to submit work and when it's safe to release application resources. In next month's column, I'll examine the thread pool environment as I continue to explore the Windows thread pool API. ■

KENNY KERR is a software craftsman with a passion for native Windows development. Reach him at kennykerr.ca.

THANKS to the following technical experts for reviewing this article:
Hari Pulapaka and Pedro Teixeira

Data Quality Tools for .NET



Clean your database with tools that make it easy.

Request a free trial at
MelissaData.com/mynet or
Call 1-800-MELISSA (635-4772)

MELISSA DATA®
Your Partner in Data Quality



Searching Windows Azure Storage with Lucene.Net

You know what you need is buried in your data somewhere in the cloud, but you don't know where. This happens so often to so many, and typically the response is to implement search in one of two ways. The simplest is to put key metadata into a SQL Azure database and then use a WHERE clause to find the Uri based on a LIKE query against the data. This has very obvious shortcomings, such as limitations in matching based only on the key pieces of metadata instead of the document content, potential issues with database size for SQL Azure, added premium costs to store metadata in SQL Azure, and the effort involved in building a specialized indexing mechanism often implemented as part of the persistence layer. In addition to those shortcomings are more specialized search capabilities that simply won't be there, such as:

- Relevance ranking
- Language tokenization
- Phrase matching and near matching
- Stemming and synonym matching

A second approach I've seen is to index the cloud content from the local search indexer, and this has its own problems. Either the document is indexed locally and the Uri fixed up, which leads to complexity in the persistence and indexing because the file must be local and in the cloud; or the local indexing service reaches out to the cloud, which decreases performance and increases bandwidth consumption and cost. Moreover, using your local search engine could mean increased licensing costs, as well. I've seen a hybrid of the two approaches using a local SQL Server and full-text Indexing.

Theoretically, when SQL Azure adds full-text indexing, you'll be able to use the first method more satisfactorily, but it will still require a fair amount of database space to hold the indexed content. I wanted to address the problem and meet the following criteria:

1. Keep costs relating to licensing, space and bandwidth low.
2. Have a real search (not baling wire and duct tape wrapped around SQL Server).
3. Design and implement an architecture analogous to what I might implement in the corporate datacenter.

Code download available at code.msdn.microsoft.com/mag201108Cloudy.

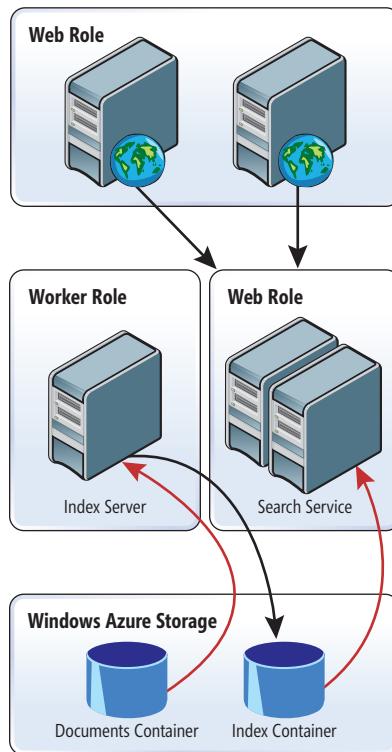


Figure 1 Search Architecture

Search Architecture Using Lucene.Net

I want a real search architecture and, thus, need a real indexing and search engine. Fortunately, many others wanted the same thing and created a nice .NET version of the open source Lucene search and indexing library, which you'll find at incubator.apache.org/lucene.net. Moreover, Tom Laird-McConnell created a fantastic library for using Windows Azure Storage with Lucene.Net; you'll find it at code.msdn.microsoft.com/AzureDirectory. With these two libraries I only need to write code to crawl and index the content and a search service to find the content. The architecture will mimic typical search architecture, with index storage, an indexing service, a search service and some front-end Web servers to consume the search service (see Figure 1).

The Lucene.Net and AzureDirectory libraries will run on a Worker Role that will serve as the Indexing Service, but the front-end Web Role only needs to consume the search service and doesn't need the search-specific libraries. Configuring the storage and compute instances in the same region should keep

bandwidth use—and costs—down during indexing and searching.

Crawling and Indexing

The Worker Role is responsible for crawling the documents in storage and indexing them. I've narrowed the scope to handle only Word .docx documents, using the OpenXML SDK 2.0, available at msdn.microsoft.com/library/bb456488. I've chosen to actually insert the latest code release for both AzureDirectory and Lucene.Net in my project, rather than just referencing the libraries.

Within the Run method, I do a full index at the start and then fire an incremental update within the sleep loop that's set up, like so:

```
Index(true);  
  
while (true)  
{  
    Thread.Sleep(18000);  
    Trace.WriteLine("Working", "Information");  
  
    Index(false);  
}
```

WINFORMS • WPF • WEBFORMS • SILVERLIGHT
MVC • ACTIVEX • COMPACT FRAMEWORK

THE GREATEST FLEXGRID ON EARTH

	Name	C	W	V	M	S
1	Line: Computers (101 items)	586,190.19	557,806.17	557,806.17	557,806.17	557,806.17
2	Color: Green (36 items)	557,806.17	557,806.17	557,806.17	557,806.17	557,806.17
3	+ Rating: 4 (5 items)	1,011,490.89 (-4.3%) ▲ ↘	36,798.16 (-2.8%) ▼ ↗	36,798.16 (-2.8%) ▼ ↗	524	524
9	+ Rating: 2 (12 items)	155,165.42 (-3.4%) ▼ ↗	396,591.65 (1.2%) ▲ ✓	396,591.65 (1.2%) ▲ ✓	275,	275,
22	+ Rating: 3 (9 items)	34,478.66 (-1.5%) ▼ ↗	501,747.60 (3.0%) ▲ ↗	501,747.60 (3.0%) ▲ ↗	268	268
32	+ Rating: 0 (6 items)	74,545.47 (3.8%) ▲ ✓	6,264.79 (3.9%) ▲ ↗	6,264.79 (3.9%) ▲ ↗	40,	40,
39	+ Rating: 1 (4 items)	31,347.88 (4.7%) ▲ ✓	2,730.39 (-4.4%) ▼ ↗	2,730.39 (-4.4%) ▼ ↗	17,	17,
44	+ Color: Blue (28 items)	79,511.15 (-0.2%) ▼ ↗	108,971.39 (-5.0%) ▼ ↗	108,971.39 (-5.0%) ▼ ↗	94,	94,
78	+ Color: Red (22 items)	42,038.36 (-3.9%) ▼ ↗	123,822.07 (3.6%) ▼ ↗	123,822.07 (3.6%) ▼ ↗	82,	82,
106	+ Color: White (15 items)	36,443.26 (2.1%) ▲ ↗	15,309.72 (2.4%) ▲ ↗	15,309.72 (2.4%) ▲ ↗	25,	25,
127	+ Line: Washers (77 items)	49,039.67 (4.6%) ▲ ↗	74,546.89 (4.8%) ▲ ↗	74,546.89 (4.8%) ▲ ↗	61,	61,
229	+ Line: Stoves (72 items)	129,389.49 (-1.0%) ▼ ↗	43,242.59 (2.3%) ▲ ↗	43,242.59 (2.3%) ▲ ↗	86,	86,
261	+ Line: Dyers (70 items)	49,039.67 (2.8%) ▲ ↗	74,546.89 (-2.3%) ▼ ↗	74,546.89 (-2.3%) ▼ ↗	61,	61,
289	+ Line: Microwaves (55 items)	29,898.06 (5.6%) ▲ ↗	11,165.38 (-2.7%) ▼ ↗	11,165.38 (-2.7%) ▼ ↗	20,	20,
321	+ Line: Toasters (82 items)	85,025.84 (-3.3%) ▼ ↗	84,515.76 (-1.7%) ▼ ↗	84,515.76 (-1.7%) ▼ ↗	84,	84,

"I LOVE, LOVE, LOVE, the FlexGrid. It's like a Swiss army knife and I will never switch!"

Darrin P. Dyson, Director of Development and Systems



LIGHTWEIGHT!
PRINTING SUPPORT!
CELL MERGING!

SUPER FLEXIBLE!
EXCEL-LIKE EDITING!
UNBOUND MODE!



Experience The Magic @
COMPONENTONE.COM/FLEX



ComponentOne®

Figure 2 Pulling Out the Contents of the .docx File

```
switch(doc.Name.Substring(doc.Name.LastIndexOf(".") + 1))
{
    case "docx":
        WordprocessingDocument wordprocessingDocument =
            WordprocessingDocument.Open(localStream, false);
        doc.Body = wordprocessingDocument.MainDocumentPart.Document.Body.InnerText;
        wordprocessingDocument.Close();
        break;
    // TODO: Still incomplete
    case "pptx":
        // Probably want to create a generic for DocToIndex and use it
        // to create a pptx-specific that allows slide-specific indexing.
        PresentationDocument pptDoc = PresentationDocument.Open(localStream, false);
        foreach (SlidePart slide in pptDoc.PresentationPart.SlideParts)
        {
            // Iterate through slides
        }
        break;
    default:
        break;
}
```

For my sample, I keep the loop going at a reasonable frequency by sleeping for 18,000 ms. I haven't created a method for triggering an index, but it would be easy enough to add a simple service to trigger this same index method on demand to manually trigger an index update from an admin console or call it from a process that monitors updates and additions to the storage container. In any case, I still want a scheduled crawl and this loop could serve as a simple implementation of it.

Within the Index(bool) method I first check to see if the index exists. If an index does exist, I won't generate a new index because it would obliterate the old one, which would mean doing a bunch of unnecessary work because it would force a full indexing run:

```
DateTime LastModified = new DateTime(IndexReader.LastModified(azureDirectory), DateTimeKind.Utc);
bool GenerateIndex = !IndexReader.IndexExists(azureDirectory) ;
DoFull = GenerateIndex;
```

Once I determine the conditions of the index run, I have to open the index and get a reference to the container that holds the documents being indexed. I'm dealing with a single container and no folders, but in a production implementation I'd expect multiple containers and subfolders. This would require a bit of looping and recursion, but that code should be simple enough to add later:

```
// Open AzureDirectory, which contains the index
AzureDirectory azureDirectory = new AzureDirectory(storageAccount, "CloudIndex");

// Loop and fetch the information for each one.
// This needs to be managed for memory pressure,
// but for the sample I'll do all in one pass.
IndexWriter indexWriter = new IndexWriter(azureDirectory, new
StandardAnalyzer(Lucene.Net.Util.
Version.LUCENE_29), GenerateIndex, IndexWriter.MaxFieldLength.UNLIMITED);

// Get container to be indexed.
CloudBlobContainer Container = BlobClient.GetContainerReference("documents");
Container.CreateIfNotExist();
```

Using the AzureDirectory library allows me to use a Windows Azure Storage container as the directory for the Lucene.Net index without having to write any code of my own, so I can focus solely on the code to crawl and index. For this article, the two most interesting parameters in the IndexWriter constructor are the Analyzer and the GenerateIndex flag. The Analyzer is what's responsible for taking the data passed to the IndexWriter, tokenizing it and creating the index. GenerateIndex is important because if it isn't set

properly the index will get overwritten each time and cause a lot of churn. Before getting to the code that does the indexing, I define a simple object to hold the content of the document:

```
public class DocumentToIndex
{
    public string Body;
    public string Name;
    public string Uri;
    public string Id;
}
```

As I loop through the container, I grab each blob reference and create an analogous DocumentToIndex object for it. Before adding the document to the index, I check to see if it has been modified since the last indexing run by comparing its last-modified time to that of the LastModified time of the Index that I grabbed at the start of the index run. I will also index it if the DoFull flag is true:

```
foreach (IListBlobItem currentBlob in Container.ListBlobs(options))
{
    CloudBlob blobRef = Container.GetBlobReference(currentBlob.Uri.ToString());
    blobRef.FetchAttributes(options);
    // Add doc to index if it is newer than index or doing a full index
    if (LastModified < blobRef.Properties.LastModifiedUtc || DoFull )
    {
        DocumentToIndex curBlob = GetDocumentData(currentBlob.Uri.ToString());

        //docs.Add(curBlob);
        AddToCatalog(indexWriter, curBlob);
    }
}
```

For this simple example, I'm checking that the last modified time of the index is less than that of the document and it works well enough. However, there is room for error, because the index could've been updated via optimization and then it would look like the index was newer than a given document, but the document did not get indexed. I'm avoiding that possibility by simply tying an optimization call with a full index run. Note that in a real implementation you'd want to revisit this decision. Within the loop I call GetDocumentData to fetch the blob and AddToCatalog to add the data and fields I'm interested in to the Lucene.Net index. Within GetDocumentData, I use fairly typical code to fetch the blob and set a couple of properties for my representative object:

```
// Stream stream = File.Open(docUri, FileMode.Open);
var response = WebRequest.Create(docUri).GetResponse();
Stream stream = response.GetResponseStream();

// Can't open directly from URI, because it won't support seeking
// so move it to a "local" memory stream
Stream localStream= new MemoryStream();
stream.CopyTo(localStream);

// Parse doc name
doc.Name = docUri.Substring(docUri.LastIndexOf("/") + 1);
doc.Uri = docUri;
```

Getting the body is a little bit more work. Here, I set up a switch statement for the extension and then use OpenXml to pull the contents of the .docx out (see **Figure 2**). OpenXml requires a stream that can do seek operations, so I can't use the response stream directly. To make it work, I copy the response stream to a memory stream and use the memory stream. Make a note of this operation, because if the documents are exceptionally large, this could theoretically cause issues by putting memory pressure on the worker and would require a little fancier handling of the blob.

My additional stub and comments show where to put the code to handle other formats. In a production implementation, I'd pull



Telerik

bringing you the best
XAML components today.

Ready for what you //build/ tomorrow.



RadControls for **Silverlight** and **WPF**

Try now at: www.telerik.com/XAMLmsdn



2011 PARTNER OF THE YEAR
Mobility
Finalist

telerik
deliver more than expected

the code out for each document type and put it in a separate library of document adapters, then use configuration and document inspection to resolve the document type to the proper adapter library. Here I've placed it right in the switch statement.

Now the populated DocumentToIndex object can be passed to AddToCatalog to get it into the index (**Figure 3**).

I decided to index three fields: Title, Uri and Body (the actual content). Note that for Title and Body I use the ANALYZED flag. This tells the Analyzer to tokenize the content and store the tokens. I want to do this for the body, especially, or my index will grow to be larger in size than all the documents combined. Take note that the Uri is set to NOT_ANALYZED. I want this field stored in the index directly, because it's a unique value by which I can retrieve a specific document. In fact, I use it in this method to create a Term (a construct used for finding documents) that's passed to the UpdateDocument method of the IndexWriter. Any other fields I might want to add to the index, whether to support a document preview or to support faceted searching (such as an author field), I'd add here and decide whether to tokenize the text based on how I plan to use the field.

Implementing the Search Service

Once I had the indexing service going and could see the segment files in the index container, I was anxious to see how well it worked. I cracked open the IService1.cs file for the search service and made some changes to the interfaces and data contracts. Because this file generates SOAP services by default, I decided to stick with those for the first pass. I needed a return type for the search results, but the document title and Uri were enough for the time being, so I defined a simple class to be used as the DataContract:

```
[DataContract]
public class SearchResult
{
    [DataMember]
    public string Title;
    [DataMember]
    public string Uri;
}
```

Using the SearchResult type, I defined a simple search method as part of the ISearchService ServiceContract:

```
[ServiceContract]
public interface ISearchService
{
    [OperationContract]
    List<SearchResult> Search(
        string SearchTerms);
}
```

Next, I opened SearchService.cs and added an implementation for the Search operation. Once again AzureDirectory comes into play, and I instantiated a new one from the configuration to pass to the IndexSearcher object. The AzureDirectory library not only provides a directory interface for Lucene.NET, it also adds an intelligent layer of indirection that caches and compresses. AzureDirectory operations

Figure 3 Passing DocumentToIndex to the AddToCatalog Method

```
public void AddToCatalog(IndexWriter indexWriter, DocumentToIndex currentDocument)
{
    Term deleteTerm = new Term("Uri", currentDocument.Uri);
    LuceneDocs.Document doc = new LuceneDocs.Document();
    doc.Add(new LuceneDocs.Field("Uri", currentDocument.Uri, LuceneDocs.Field.Store.YES,
        LuceneDocs.Field.Index.NOT_ANALYZED, LuceneDocs.Field.TermVector.NO));
    doc.Add(new LuceneDocs.Field("Title", currentDocument.Name, LuceneDocs.Field.Store.YES,
        LuceneDocs.Field.Index.ANALYZED, LuceneDocs.Field.TermVector.NO));
    doc.Add(new LuceneDocs.Field("Body", currentDocument.Body, LuceneDocs.Field.Store.YES,
        LuceneDocs.Field.Index.ANALYZED, LuceneDocs.Field.TermVector.NO));
    indexWriter.UpdateDocument(deleteTerm, doc);
}
```

happen locally and writes are moved to storage upon being committed, using compression to reduce latency and cost. At this point a number of Lucene.NET objects come into play. The IndexSearcher will take a Query object and search the index. However, I only have a set of terms passed in as a string. To get these terms into a Query object I have to use a QueryParser. I need to tell the QueryParser what fields the terms apply to, and provide the terms. In this implementation I'm only searching the content of the document:

```
// Open index
AzureDirectory azureDirectory = new AzureDirectory(storageAccount, "cloudindex");
IndexSearcher searcher = new IndexSearcher(azureDirectory);

// For the sample I'm just searching the body.
QueryParser parser = new QueryParser("Body", new StandardAnalyzer());
Query query = parser.Parse("Body:( " + SearchTerms + " )");
Hits hits = searcher.Search(query);
```

If I wanted to provide a faceted search, I would need to implement a means to select the field and create the query for that field, but I'd have to add it to the index in the previous code where Title, Uri and Body were added. The only thing left to do in the service now is to iterate over the hits and populate the return list:

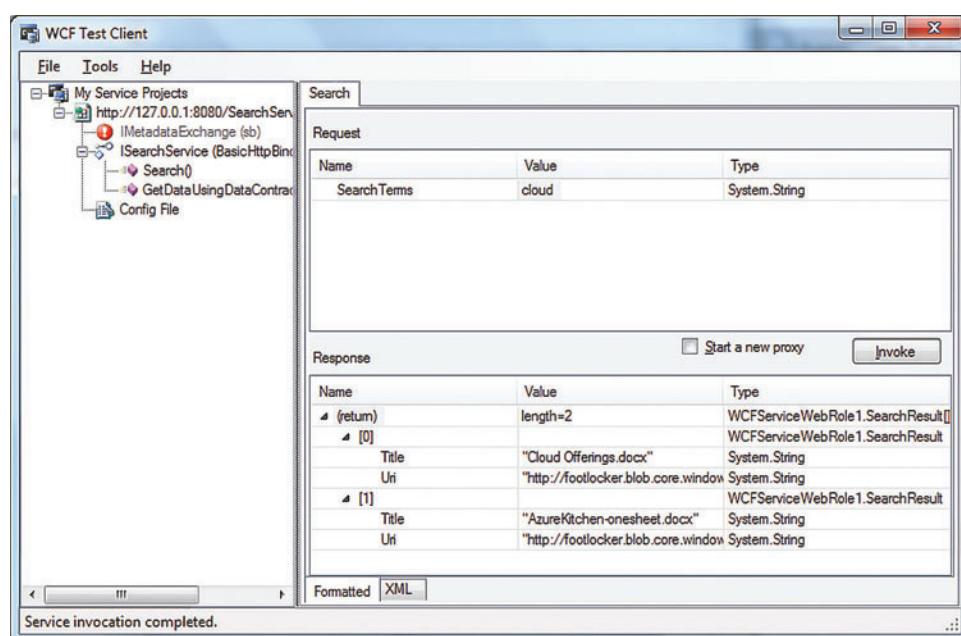


Figure 4 WcfTestClient Results

Syncfusion: Facts in Black & White

We concentrate on stability and reliability. It is no secret. With a quiet commitment to customers and code, you may not be aware that we're the backbone of many applications. The reason: While others concentrate on flash-in-the-pan, we put sustenance on the table.



Time Tested

We've been with .NET since its beginning, and in that sense, we're proud to have some age on us. It means that our controls have been brought to maturity. Stability and reliability are their earmarks. When you drop one of our components in your application, you can be confident it's been tested not only by our QA, but also within the context of the real world.



Vast Offering

We've created one of the largest arrays of components on the market. Count the number of discrete controls, and you'll see. The breadth of our offering provides the flexibility to change all aspects of your application fully without relearning the differing minds of multiple vendors. With Syncfusion, all you need to know is one.



Forward Looking

With an eye on the horizon, we advance in HTML5, ASP.NET MVC, and Silverlight for web and Windows Phone 7. We possess a passion for the new. A thirst for progress. And we invite you to come see all are doing, and all we will do, in the new frontiers of development.

what we mean by 360° transformation

Syncfusion is your development partner—a helping hand outreached. In one ad, we couldn't possibly sell you on all our products; with one magazine, we couldn't possibly convince you of our commitment to service. But we can show you. We invite you to test our full 30-day evaluation—we are confident it is enough. During the trial, please utilize our support to see what we mean by 360° transformation.

Download at www.syncfusion.com/downloads/evaluation

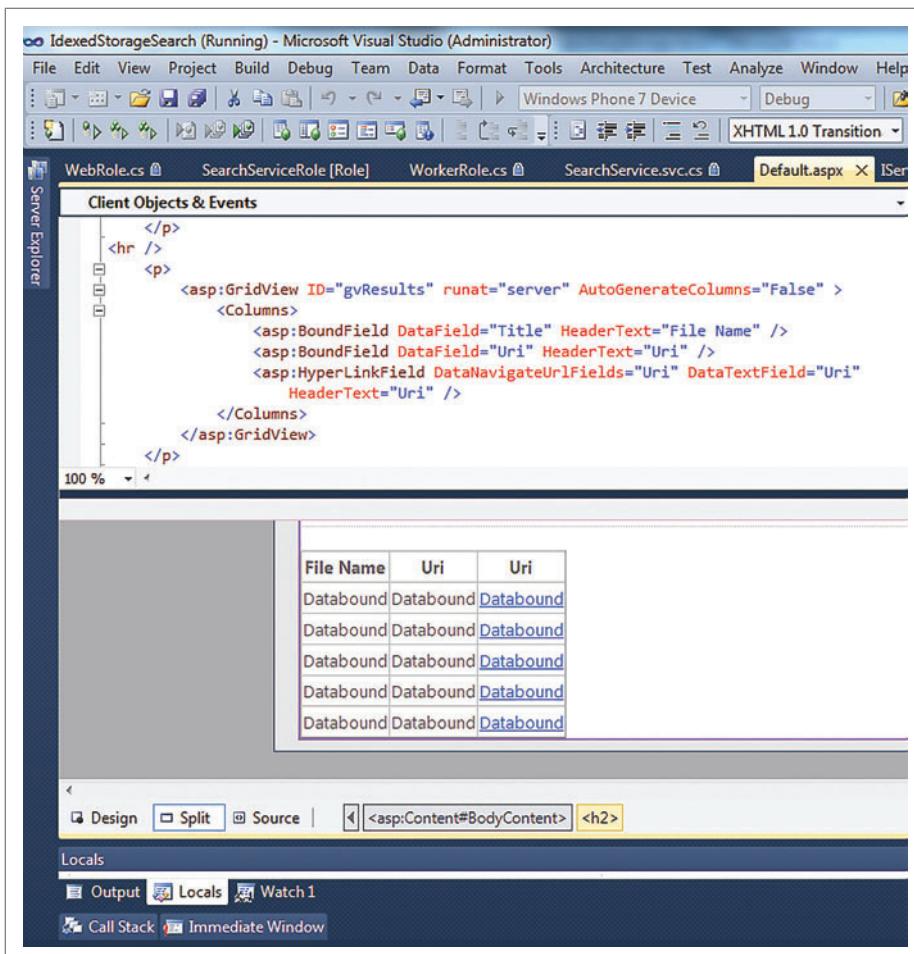


Figure 5 Defining Databound Columns for the Data Grid

```
for (int idxResults = 0; idxResults < hits.Length(); idxResults++)
{
    SearchResult newSearchResult = new SearchResult();
    Document doc = hits.Doc(idxResults);

    newSearchResult.Title = doc.GetField("Title").StringValue();
    newSearchResult.Uri = doc.GetField("Uri").StringValue();
    retval.Add(newSearchResult);
}
```

Because I'm a bit impatient, I don't want to wait to finish the Web front end to test it out, so I run the project, fire up WcfTestClient, add a reference to the service and search for the term "cloud" (see Figure 4).

I'm quite happy to see it come back with the expected results.

File Name	Uri	Uri
AzureKitchen-onesheet.docx	http://footlocker.blob.core.windows.net/documents/AzureKitchen-onesheet.docx	http://footlocker.blob.core.windows.net/documents/AzureKitchen-onesheet.docx
Cloud Offerings.docx	http://footlocker.blob.core.windows.net/documents/Cloud Offerings.docx	http://footlocker.blob.core.windows.net/documents/Cloud Offerings.docx

Figure 6 Search Results

Note that while I'm running the search service and indexing roles from the Windows Azure compute emulator, I'm using actual Windows Azure Storage.

Search Page

Switching to my front-end Web role, I make some quick modifications to the default.aspx page, adding a text box and some labels.

As Figure 5 shows, the most significant mark-up change is in the data grid where I define databound columns for Title and Uri that should be available in the resultset.

I quickly add a reference to the search service project, along with a bit of code behind the Search button to call the search service and bind the results to the datagrid:

```
protected void btnSearch_Click(
    object sender, EventArgs e)
{
    refSearchService.SearchServiceClient
        searchService = new
        refSearchService.SearchServiceClient();
    IList<SearchResult> results =
        searchService.Search(
            txtSearchTerms.Text);
    gvResults.DataSource = results;
    gvResults.DataBind();
}
```

Simple enough, so with a quick tap of F5 I enter a search term to see what I get. Figure 6 shows the results. As expected, entering "neudesic" returned

two hits from various documents I had staged in the container.

Final Thoughts

I didn't cover the kind of advanced topics you might decide to implement if the catalog and related index grow large enough. With Windows Azure, Lucene.Net and a sprinkle of OpenXML, just about any searching requirements can be met. Because there isn't a lot of support for a cloud-deployed search solution yet, especially one that could respect a custom security implementation on top of Windows Azure Storage, Lucene.Net might be the best option out there as it can be bent to fit the requirements of the implementer. ■

JOSEPH FULTZ is a software architect at AMD, helping to define the overall architecture and strategy for portal and services infrastructure and implementations. Previously he was a software architect for Microsoft working with its top-tier enterprise and ISV customers defining architecture and designing solutions.

THANKS to the following technical expert for reviewing this article:
Tom Laird-McConnell

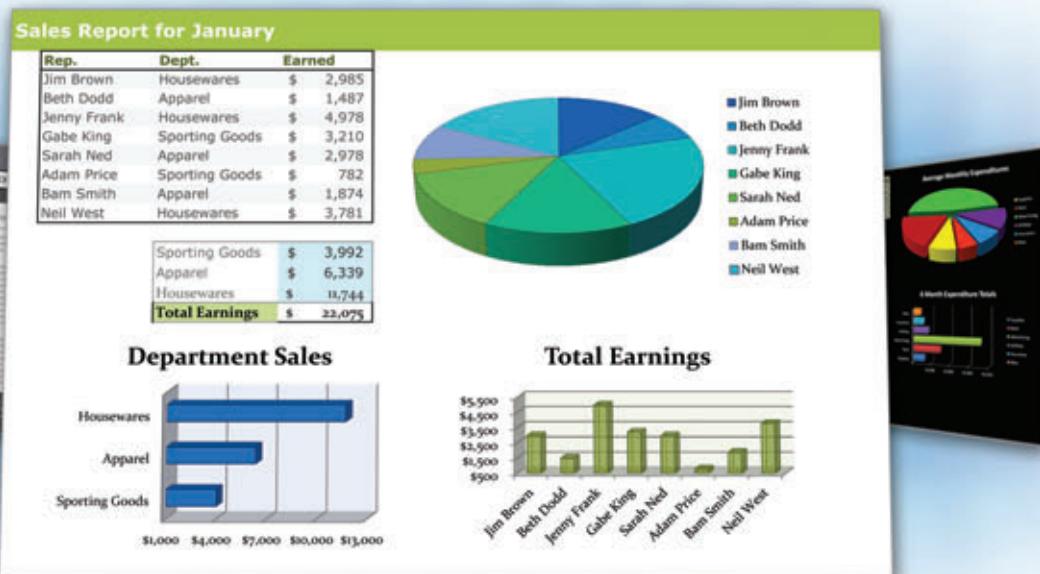
CREATING EDITING PRINTING CONVERTING REPORTING ? IMPORTING or EXPORTING ?

Documents	Spreadsheets	Presentations	Barcodes	Email
DOC DOCX	XLS HTML	PPT POTX	BMP JPG PNG	MSG
PDF HTML	TAB CSV	POT PPS	Supports 30+	MHT
TXT RTF	XLSX PDF	PPTX PDF	Symbologies	EML

...and many more file formats

100% Standalone - No Automation!

.Net Java Sharepoint SQL Reporting JasperReports



9,500+ Customers 37,500+ Licenses 130K+ User Community

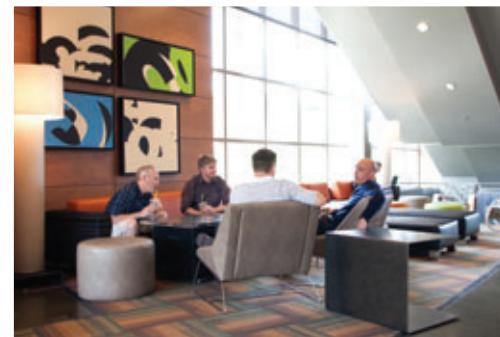


Get your FREE evaluation copy at <http://www.aspose.com>.

US Sales: 1.888.277.6734 . sales@aspose.com . EU Sales: +44 (0)800 098 8425 . sales.europe@aspose.com
Enterprise Sales – enterprise.sales@aspose.com

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS



5 DAYS OF TRAINING @ MICROSOFT HEADQUARTERS!

SUPPORTED BY:

Microsoft

msdn

Microsoft Visual Studio

Visual Studio MAGAZINE

IF YOU'RE LOOKING FOR HARD-HITTING .NET DEVELOPER TRAINING,

look no further than Visual Studio Live! Redmond. Join industry pros and Microsoft insiders for relevant, practical and up-to-date education that will solve your everyday development challenges.

Plus, experience life on Campus like a blue badge!

Get employee-level access to the company store, lunch in the Microsoft "town center," and more.

PICK YOUR
SESSIONS

SAVE BIG
WHEN YOU
REGISTER NOW!
Use Priority Code: AUGAD

EVENT SPONSOR

Microsoft

PLATINUM SPONSORS

esri VERSANT

GOLD SPONSOR

DevExpress™
Download • Compare • Decide!

PRODUCED BY:

1105 MEDIA

VSLIVE.COM/REDMOND

AGENDA

OCTOBER 17-21
REDMOND, WASHINGTON

VISUAL STUDIO LIVE! REDMOND

Silverlight/WPF	Developing Services	Visual Studio 2010/.NET 4	Cloud	Data Management	LightSwitch	Programming Practices	Web/HTML5	Mobile Dev
-----------------	---------------------	---------------------------	-------	-----------------	-------------	-----------------------	-----------	------------

Visual Studio Live! Pre-Conference Workshops: Monday, October 17, 2011

MWK1 Workshop: ALM in 2011: Visual Studio 2010 and the Next Big Release *Brian Randell*

MWK2 Workshop: Making Effective Use of Silverlight and WPF *Billy Hollis & Rockford Lhotka*

MWK3 Workshop: Programming with WCF in One Day *Miguel Castro*

Visual Studio Live! Day 1: Tuesday, October 18, 2011

T1 Microsoft Session—Details TBA	T2 Intense Intro to Silverlight <i>Billy Hollis</i>	T3 AppFabric, Workflow and WCF—the Next-Generation Middleware <i>Ron Jacobs</i>	T4 If not IaaS, When Should I Use Windows: Windows Azure VM Role? <i>Eric D. Boyd</i>	T5 Best Kept Secrets in Visual Studio 2010 and .NET 4 <i>Deborah Kurata</i>
T6 Microsoft Session—Details TBA	T7 XAML: Achieving Your Moment of Clarity <i>Miguel Castro</i>	T8 What's New in WCF 4 <i>Ido Flatow</i>	T9 What Is Microsoft Marketplace DataMarket? <i>Michael Stiefel</i>	T10 The LINQ Programming Model <i>Marcel de Vries</i>
T11 Microsoft Session—Details TBA	T12 Fundamental Design Principles for UI Developers <i>Billy Hollis</i>	T13 Creating Scalable State Full Services Using WCF and WF <i>Marcel de Vries</i>	T14 Deciding Between Relational Databases and Tables in the Cloud <i>Michael Stiefel</i>	T15 NoSQL—Beyond the Key-Value Store <i>Robert Green</i>
T16 HTML5 and Internet Explorer 9: Developer Overview <i>Ben Hoelting</i>	T17 Bind Anything to Anything in XAML <i>Rockford Lhotka</i>	T18 AppFabric Caching: How It Works and When You Should Use It <i>Jon Flanders</i>	T19 Microsoft Session—Details TBA	T20 How to Take WCF Data Services to the Next level <i>Rob Daigneau</i>

Microsoft Ask the Experts & Welcome Reception

Visual Studio Live! Day 2: Wednesday, October 19, 2011

W1 HTML5 and Your Web Sites <i>Robert Boedigheimer</i>	W2 Handling Offline Data in Silverlight and Windows Phone 7 <i>John Papa</i>	W3 Microsoft Session—Details TBA	W4 Windows Azure Platform Overview <i>Vishwas Lele</i>	W5 ALM <i>Brian Randell</i>
W6 Styling Web Pages with CSS 3 <i>Robert Boedigheimer</i>	W7 Building Great Windows Applications with XAML and C# LINQ <i>Pete Brown</i>	W8 Building Native Mobile Apps with HTML5 & jQuery <i>Jon Flanders</i>	W9 Building Windows Azure Applications <i>Vishwas Lele</i>	W10 Visual Studio <i>Brian Randell</i>
W11 The Best of jQuery <i>Robert Boedigheimer</i>	W12 What's New and Cool in Silverlight 5 <i>Pete Brown</i>	W13 Getting Started with Windows Phone 7 <i>Scott Golightly</i>	W14 Building Compute-Intensive Apps in Windows Azure <i>Vishwas Lele</i>	W15 Bringing Your Data and Maps Together with Esri Cloud Services <i>Arthur J. Haddad</i>
W16 ASP.NET MVC, Razor and jQuery—the New Face of ASP.NET <i>Ido Flatow</i>	W17 Silverlight, WCF, RIA Services and Your Business Objects <i>Deborah Kurata</i>	W18 Windows Azure and Windows Phone—Creating Great Apps <i>Scott Golightly</i>	W19 Building and Running the Windows Azure Developer Portal <i>Chris Mullins</i>	W20 Microsoft Session—Details TBA

Sponsor Reception / Wild Wednesday

Visual Studio Live! Day 3: Thursday, October 20, 2011

TH1 WebMatrix and Razor <i>Rachel Appel</i>	TH2 Bringing the Silverlight PivotViewer to Your Applications <i>Tony Champion</i>	TH3 CSLA 4 for Windows Phone and Silverlight <i>Rockford Lhotka</i>	TH4 Microsoft Session—Details TBA	TH5 Design for Testability: Mocks, Stubs, Refactoring and UIs <i>Ben Day</i>
TH6 Orchard <i>Rachel Appel</i>	TH7 MVVM in Practice aka "Code Behind"—Free WPF <i>Tiberiu Covaci</i>	TH8 Working with Data on Windows Phone 7 <i>Sergey Barskiy</i>	TH9 Microsoft Session—Details TBA	TH10 Team Foundation Server 2010 Builds: Understand, Configure and Customize <i>Ben Day</i>
TH11 Busy Developer's Guide to (ECMA/Java)Script <i>Ted Neward</i>	TH12 Radically Advanced Templates for WPF and Silverlight <i>Billy Hollis</i>	TH13 Advanced Patterns with MVVM in Silverlight and Windows Phone 7 <i>John Papa</i>	TH14 Microsoft Session—Details TBA	TH15 LightSwitch 202 <i>Andrew Brust</i>
TH16 Getting Started with ASP.NET MVC <i>Philip Japikse</i>	TH17 Using MEF to Develop Composable Applications <i>Ben Hoelting</i>	TH18 Windows Phone 7 Instrumentation—How to Learn from Your App <i>Tony Champion</i>	TH19 So Many Choices, So Little Time: Understanding Your .NET 4 Data Access Options <i>Lenni Lobel</i>	TH20 Static Analysis in .NET <i>Jason Bock</i>
TH21 Test Driving ASP.NET MVC <i>Philip Japikse</i>	TH22 Patterns for Parallel Programming <i>Tiberiu Covaci</i>	TH23 Microsoft Session—Details TBA	TH24 Using Code First (Code Only) Approach with Entity Framework <i>Sergey Barskiy</i>	TH25 Modern .NET Development Practices and Principles <i>Jason Bock</i>

Visual Studio Live! Post-Conference Workshops: Friday, October 21, 2011

FWK1 Architectural Katas Workshop *Ted Neward*

FWK2 SQL Server Workshop for Developers *Andrew Brust & Leonard Lobel*

VISIT US ONLINE AT VSLIVE.COM/REDMOND FOR MORE DETAILS!

Sessions and speakers are subject to change.

Advanced Programming Made Easy with Visual Studio LightSwitch

Beth Massi

Visual Studio LightSwitch is a new product in the Visual Studio family aimed at developers who want to quickly create data-centric business applications for the desktop and the cloud. LightSwitch is an extensible development environment that simplifies the development process because it lets you concentrate on the business logic while it takes care of a lot of the remaining work. LightSwitch is perfect for small business or departmental productivity applications that need to be done *fast*.

It's easy to quickly get up and running in LightSwitch so you can focus on features important to your business. You don't necessarily need to write code to get a LightSwitch application up and running, but you'll quickly realize that code is necessary for business rules, screen workflows and other user productivity features specific to your business requirements. Moreover, you can download and install LightSwitch extensions from the Visual Studio Gallery that add features to your application that aren't available out of the

box. You can even create extensions yourself using Visual Studio Professional or higher. It's a good bet that the community will be creating some interesting extensions for LightSwitch!

This article is aimed at the professional developer using LightSwitch to speed up development of a typical data-centric business application, or enhancing a LightSwitch application that was initially built by someone else. I'm going to walk through some of the more advanced development and customization techniques available to LightSwitch developers. I'll show you how to work with LightSwitch APIs, create custom screen layouts, use advanced query processing, write complex business rules, and use and create LightSwitch extensions. You'll see that programming even the more advanced features of a LightSwitch business application is simplified dramatically because LightSwitch handles all the plumbing for you. You can download the sample application I'll be discussing at code.msdn.microsoft.com/Contoso-Construction-9f944948.

If you're not familiar with creating a basic application using LightSwitch, I encourage you to take a look at some of the resources available on the LightSwitch Developer Center at msdn.com/lightswitch.

This article discusses:

- LightSwitch application architecture
- LightSwitch extensibility points
- Screen layout and templates
- Adding custom code to the client
- Working with the save pipeline
- Using and building extensions

Technologies discussed:

Visual Studio LightSwitch, Visual Studio, Silverlight,
Microsoft .NET Framework 4

Code download available at:

code.msdn.microsoft.com/Contoso-Construction-9f944948

Architecture Overview

Before we dive into advanced LightSwitch development, it's important to understand the architecture of a LightSwitch application. LightSwitch applications are based on Silverlight and a solid .NET application framework using well-known patterns and best practices, like *n*-tier application layering and Model-View-ViewModel (MVVM), as well as technologies like the Entity Framework and WCF RIA Services (see **Figure 1**). The LightSwitch team made sure not to invent new core technologies, such as a new data-access or UI layer; instead we created an application framework and development environment around these existing Microsoft .NET Framework-based technologies that lots of developers are already building on today.

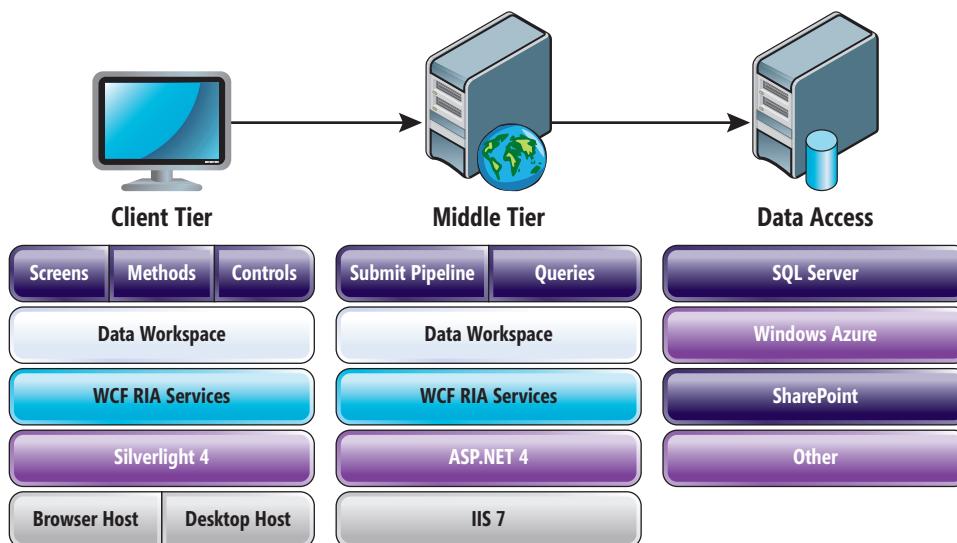


Figure 1 LightSwitch Application Architecture

This architecture means LightSwitch applications can be deployed as desktop applications, with the ability to integrate with hardware devices such as a scanner or bar code reader, as well as other applications like Microsoft Word and Excel; or they can be deployed as browser-based applications when broader reach is required. You can also host LightSwitch applications in a variety of ways, including with Windows Azure.

LightSwitch is all about data and screens. The data in a screen is managed by a data workspace, which is responsible for fetching entities from a data service in the middle tier via queries, for tracking changes, and for sending updates back through the save pipeline to the middle-tier data service. When you execute the save command, it invokes the data service with one change set, which runs in one update transaction against the data source. You can have multiple data sources in LightSwitch (and even relate entities across data sources). Each data source you bring into your LightSwitch application has its own data workspace. From the data workspace, you can access all the queries and update operations against the data source. Entities themselves also expose underlying details, such as their entity state and original and current values.

Visual Studio LightSwitch is its own edition of Visual Studio. Its entire development environment is streamlined to focus on building LightSwitch applications. Even with just the LightSwitch edition installed, developers have access to all the LightSwitch APIs, the Silverlight and .NET frameworks, custom screen layouts and screen workflows, complex validation, the save pipeline and access control hooks. You can write complex and composite LINQ

queries and use COM interop, as well as access both client and server project code.

However, if you already have Visual Studio Professional or higher installed, LightSwitch will integrate into those editions and LightSwitch applications will show up as just another project template in the New Project dialog. In addition, you also have the ability to create custom controls and extensions for LightSwitch.

Customization and Extensibility Points

LightSwitch allows a wide spectrum of customization, from simple code customization all the way to authoring full-blown extensions

and distributing them to other LightSwitch developers.

There are a lot of built-in hooks to write code in LightSwitch for entities, queries and screens. At the top of each designer you'll see a "Write Code" drop-down button that shows the method hooks available. This is how you write validation and security rules, as well as how you tap into the save pipeline and query processing. Some code you write runs on the server, some on the client and some runs on both. Typically LightSwitch developers don't need to know where their code is running; LightSwitch takes care of all of that for you. However, if you want to provide additional functionality and need to write specific code on the client or the server, you need to know a little bit about the structure of a LightSwitch solution.

When you look at a LightSwitch application in the Solution Explorer, by default you're in "Logical View." However, you can see the actual project and file structure if you select "File View," as shown in **Figure 2**.

In File View you can see the Client and Server projects and, under those, a UserCode folder where all the code you write through the LightSwitch development environment is physically placed. You can add your own code files and classes here and call them from your LightSwitch code. Just keep in mind that the client is running under the Silverlight framework and the server is running under the full .NET Framework 4. I'll show you a couple of examples later.

The next level of customization is authoring custom controls and custom data sources and using them directly in your LightSwitch project. For instance, you may choose to create your own special Silverlight control that provides

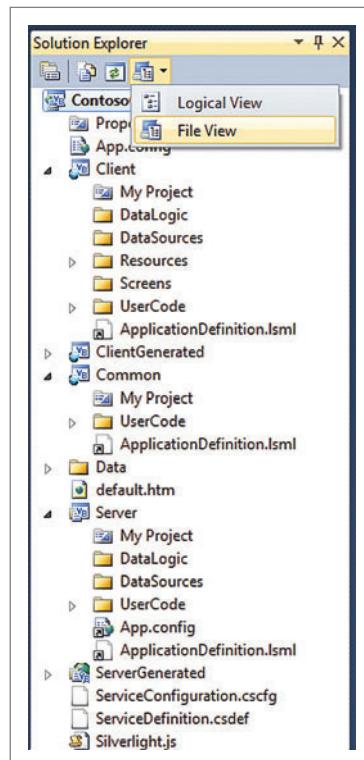


Figure 2 Flip to File View to See the Physical Structure

some specific functionality and then use that on your LightSwitch screens. If you have Visual Studio Professional or higher, you have access to the Silverlight class library project template and you can simply build your own control library and use it in LightSwitch. You can also create WCF RIA Services to connect to data sources not supported out of the box in LightSwitch. However, in order to make these custom controls and data sources available to someone who only has the Visual Studio LightSwitch edition installed, you package them up into a LightSwitch extension.

There are six extensibility points in LightSwitch:

1. **Custom Control** A control extension consists of one or more Silverlight controls, which can bind to a single scalar value or a collection of data. It can also group other controls together. A control extension can be a single control; for example, a star-rating control or gauge or even a map control. It could also be a group of controls; for example, shipping information.
2. **Screen Template** When creating a screen, you're presented with a list of templates. A screen template extension creates a new template that can be added to the list. This allows a custom screen to specify a layout for the controls contained within the screen.
3. **Business Type** Business types are unique to LightSwitch; they allow you to provide canned validation and formatting for entity properties. When you build this type of extension, you define an editor control and formatting as well as validation for the type. For instance, Phone Number and Email Address are built-in business types in LightSwitch and can be configured declaratively via the properties window. You could create a business type for Social Security Number or Packaging SKU that comes with specific validation and formatting.

4. **Shell** A shell provides the look and feel of the app—its “skin.” It can also provide navigation, commands and screens. It's totally up to you to decide how to use the shell to present these—or not present them.

5. **Theme** Themes provide the color palette for a LightSwitch application. The font styles and brush colors you specify in a theme are applied to the built-in shell, or to a custom shell you build yourself.

6. **Custom Data Sources** All applications need some kind of data source. LightSwitch lets you connect to databases, SharePoint lists and WCF RIA Services. With this extension type, you can connect to external data sources using WCF RIA Services.

To build extensions, you need Visual Studio Professional or higher, the Visual Studio SDK and the LightSwitch Extensibility Toolkit. Extensions are distributed just like any other Visual Studio extension via VSIX packages, and deployed to the Visual Studio Gallery. You can then install them using the Extension Manager, which is also included with Visual Studio LightSwitch.

Now that you understand the architecture, solution structure and spectrum of extensibility in LightSwitch applications, let's dive into some advanced features of the Contoso Construction sample application. I built the entire application without any specific custom control or data source customizations, so it requires only Visual Studio LightSwitch. However, I did install and use publicly available extensions to enhance the application, and I'll walk through how they work later.

The Contoso Construction Application

When you log into the Contoso Construction application, the first thing you're presented with is a tailored home screen that displays common tasks, your appointments for the day and current projects,

as shown in **Figure 3**. The application tracks customers and their construction projects as well as materials used on the projects. It also allows users to manage photos taken for each project and to schedule appointments between employees and customers.

When you select a project, the Project Details screen opens, displaying a tab control with its related children, the project's materials and pictures. Saving pictures into the database with LightSwitch is handled automatically when you specify that a property of an entity be of the built-in type Image—which comes with a picture editor and display.

The Project Details screen also has a button on the ribbon to generate a project status report. It uses COM to automate Microsoft Word to send the project and materials list to the document. Other features of

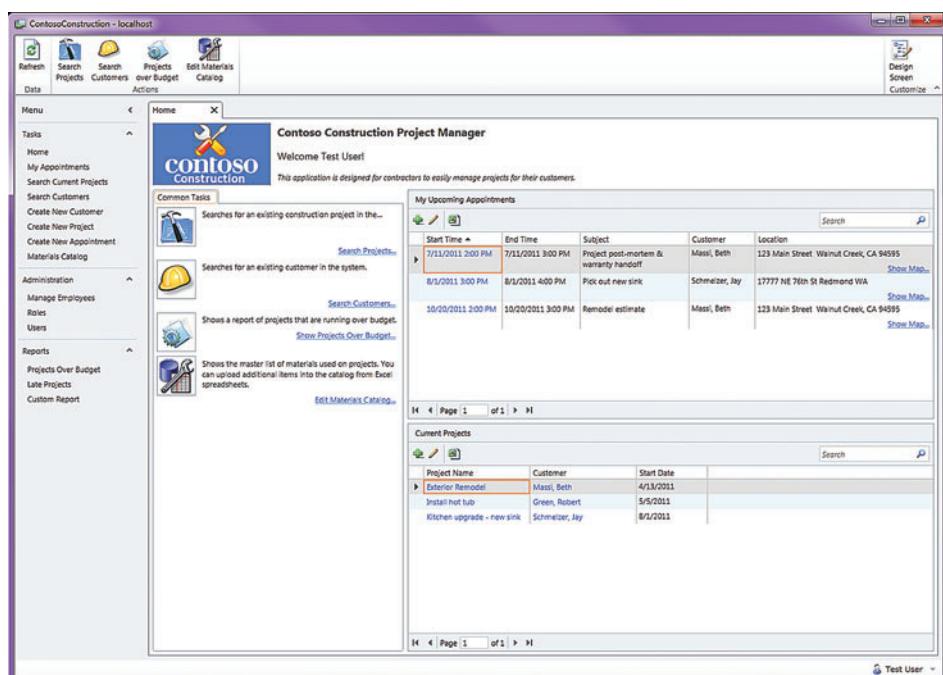


Figure 3 The Contoso Construction Application Home Screen

THESE GUYS STAND ON THEIR OWN.

That's because we focus on the most important controls, not dozens of generic, bundled ones.

The image shows three tablets side-by-side, each displaying a different application built using Xceed's controls:

- Top Tablet:** Shows a WPF application with a DataGrid control. The grid has columns for ID, Employee, Country, and Customer. It displays data for the USA and Albuquerque regions, with many rows showing the same customer information.
- Middle Tablet:** Shows a Silverlight application titled "MOVIEPICKER". It lists movie reviews for 2011, including "The Green Hornet", "The Mechanic", "Haywire", and "Sanctum". Each movie entry includes a thumbnail, title, a short description, and a rating.
- Bottom Tablet:** Shows a Silverlight application with a DataGrid control. The grid has columns for Order ID, Country, Region, City, Address, and Postal Code. It displays data for the USA, specifically for Florida (FL), showing multiple entries for Jacksonville and Miami.



XCEED
DataGrid
for WPF

Mature, feature-packed, and lightning-fast. The most adopted and trusted WPF datagrid around!



XCEED
DataGrid
for Silverlight

The only Silverlight datagrid on the market with fast remote data retrieval and a completely fluid UI!



XCEED
**Ultimate
ListBox**
for Silverlight

The same data virtualization and smooth-scrolling as our Silverlight datagrid, packed in the streamlined format of a listbox.

Try them live at xceed.com

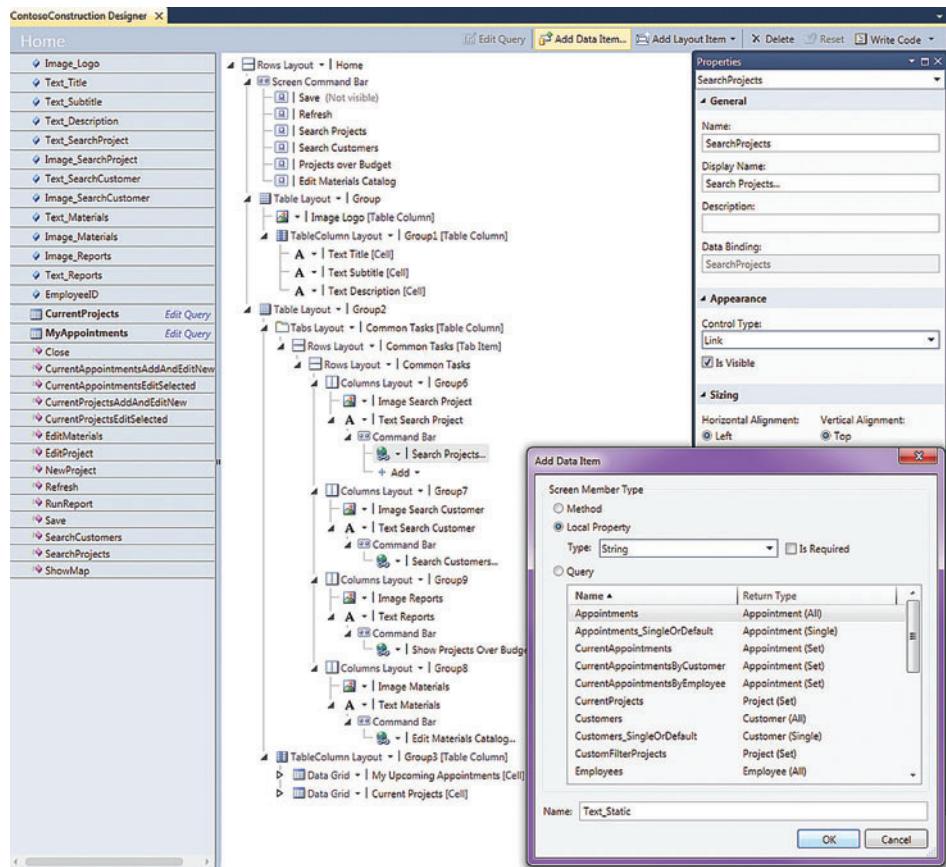


Figure 4 Data Items and the Content Tree in Screen Designer

the application include e-mail integration on the server to send appointments to customers and employees, audit trails, reports with aggregate queries, and a custom query builder. LightSwitch already has built-in support for exporting data in grids to Excel. The Contoso Construction application also has an import feature that lets users upload materials data from spreadsheets directly into the screen. Let's walk through some of the more advanced pieces of the application to see how they were built.

Complex Screen Layouts

When building screens in LightSwitch, every control in the content tree must be bound to a data item. The data item can be a field on an entity, a collection of entities or any other data item you add to the ViewModel. The screen templates help guide you in setting up a layout and all the data binding for a particular type of screen—be that a search screen, a details screen, a new data screen or any other type of screen. Some screens work with single entities and others work with collections of entities. The Search Screen, Editable Grid Screen, and List and Details Screen templates work with collections of entities returned from a query that you can define. The Edit Details Screen works with a specific entity and it passes a parameter into the query that's the unique ID of the entity to retrieve.

These templates are just starting points that you can customize further. In fact, you can select a screen template that works with a collection of entities, but not choose any data for the screen. For instance, in the Add New Screen dialog, you can select the Search Screen

and then immediately click OK. This creates a blank “canvas” that allows you to start from scratch. Alternatively, when you select a template you can select the basic screen data you want to present and then add more data items to the screen as needed.

Data items can be basic properties like strings or integers; they can be methods you call from screen code or buttons; or they can be entities that originate from queries. To add a data item manually to a screen, click the Add Data Item button at the top of the screen designer. You can then specify the name and type of the data item. The home screen is a combination of all of these types of data items, as you can see in **Figure 4**.

You have a lot of flexibility with screens when laying out content in the tree. Once you add all of your data items, in order to make layout changes it's usually easier to just run the application (press F5), open the screen and then click Design Screen in the upper-right corner of the application. This allows you to make layout changes and see the updates in

real time. You can use grouping controls like the Table Layout and Rows and Columns Layouts to fine-tune exactly where and how controls should display. You can then tweak the sizing properties in the Properties window to specify the exact size. In addition to the screen command bar (which displays on the ribbon across the top of the application), you can also specify command bars for any

Figure 5 The MyImageHelper Class

```

''' <summary>
''' This class makes it easy to load images from the client project.
''' </summary>
''' <remarks></remarks>
Public Class MyImageHelper
    Public Shared Function GetImageByName(fileName As String) As Byte()
        Dim assembly As Reflection.Assembly =
            Reflection.Assembly.GetExecutingAssembly()
        Dim stream As Stream = assembly.GetManifestResourceStream(fileName)
        Return GetStreamAsByteArray(stream)
    End Function

    Private Shared Function GetStreamAsByteArray(
        ByVal stream As System.IO.Stream) As Byte()
        If stream Is Nothing Then
            Dim streamLength As Integer = Convert.ToInt32(stream.Length)
            Dim fileData(streamLength - 1) As Byte
            stream.Read(fileData, 0, streamLength)
            stream.Close()
            Return fileData
        Else
            Return Nothing
        End If
    End Function
End Class

```

Our Name Says It All

activePDF®



Come and see why thousands of customers have trusted us over the last decade for all their server based PDF development needs.

- . Convert over 400 file types to PDF
- . High fidelity translation from PDF to Office
- . ISO 32000 Support including PDF/X & PDF/A
- . HTML5 to PDF
- . True PDF Print Server
- . Form Fill PDF
- . Append, Stamp, Secure and Digitally Sign



Microsoft Partner
Silver Midmarket Solution Provider



Download our FREE evaluation from
www.activepdf.com/MSDN

Call 1-866-GoTo-PDF | 949-582-9002 | Sales@activepdf.com

Figure 6 Server-Side Methods to Track and Send Appointment E-mails

```

Private Sub Appointments_Inserting(ByVal entity As Appointment)
    'Used to track any iCalendar appointment requests
    entity.MsgID = Guid.NewGuid.ToString()
    entity.MsgSequence = 0
End Sub

Private Sub Appointments_Updating(ByVal entity As Appointment)
    'Update the sequence anytime the appointment is updated
    entity.MsgSequence += 1
End Sub

Private Sub Appointments_Inserted(ByVal entity As Appointment)
    'Send an email notification when an appointment is inserted into the system
    Dim isCanceled = False
    SMTPMailHelper.SendAppointment(entity.Employee.Email,
                                    entity.Customer.Email,
                                    entity.Subject,
                                    entity.Notes,
                                    entity.Location,
                                    entity.StartTime,
                                    entity.EndTime,
                                    entity.MsgID,
                                    entity.MsgSequence,
                                    isCanceled)
End Sub

Private Sub Appointments_Updated(ByVal entity As Appointment)
    'Send an email update when an appointment is updated.
    Dim isCanceled = False
    SMTPMailHelper.SendAppointment(entity.Employee.Email,
                                    entity.Customer.Email,
                                    entity.Subject,
                                    entity.Notes,
                                    entity.Location,
                                    entity.StartTime,
                                    entity.EndTime,
                                    entity.MsgID,
                                    entity.MsgSequence,
                                    isCanceled)
End Sub

Private Sub Appointments_Deleting(entity As Appointment)
    'Send an email cancellation when an appointment is deleted.
    Dim isCanceled = True
    SMTPMailHelper.SendAppointment(entity.Employee.Email,
                                    entity.Customer.Email,
                                    entity.Subject,
                                    entity.Notes,
                                    entity.Location,
                                    entity.StartTime,
                                    entity.EndTime,
                                    entity.MsgID,
                                    entity.MsgSequence,
                                    isCanceled)
End Sub

```

control. This gives you the ability to place buttons or hyperlinks under any item in the tree. In **Figure 4** you can see that the command bar for the static text property called Search Projects has its Control Type set to Link, which appears left-justified under the text when running (**Figure 3**).

The home screen displays a lot of static text and images. Adding images to buttons on screens is as easy as setting them in the Properties window. However, when you want to display an image or static text on the screen directly, you need to do a couple of things. You add static images and text to the screen as local properties via the Add Data Item dialog and then lay them out in the content tree. Because static properties do not originate from a data entity, you need to set the property value before the screen is displayed. You do this in the screen's InitializeDataWorkspace method, which runs

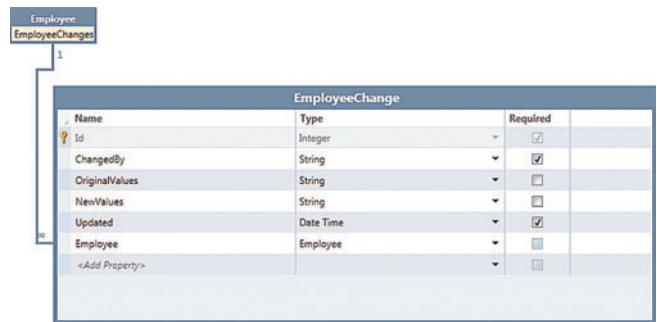


Figure 7 An Audit Trail Is a Common Business Application Requirement

before any queries execute. For example, to set one image and one text static property, you'd write code similar to this:

```

Private Sub Home_InitializeDataWorkspace(saveChangesTo As List(Of IDataService))
    ' Initialize text properties
    Text_Title = "Contoso Construction Project Manager"

    ' Initialize image properties
    Image_Logo = MyImageHelper.GetImageByName("logo.png")

```

End Sub

In order to load a static image, you switch to File View and place it in the Client project's \Resources folder, then set the build action to "Embedded Resource." Then you need to write some code to load the image. The Contoso Construction sample application uses static images in a variety of screens, so I created a helper class called MyImageHelper that can be used from anywhere in the client code. While in File View, right-click on the \UserCode folder in the Client project and select Add | Class. Name it MyImageHelper and create a static (Shared) method that loads the image, as shown in **Figure 5**.

You can add any custom code to the client this way. For instance, the Contoso Construction sample application also has helper

Figure 8 Drilling into the Employee Entity Details Property to Create an Audit Trail

```

Private Sub Employees_Updating(entity As Employee)
    'Audit trail that tracks changes to employee records
    Dim change = entity.EmployeeChanges.AddNew()
    change.Employee = entity
    change.Updated = Now()
    change.ChangedBy = Me.Application.User.FullName
    Dim newvals = "New Values:"
    Dim oldvals = "Original Values:"

    For Each prop In entity.Details.Properties.All()
        If prop.Name <> "Id" Then
            If Not Object.Equals(prop.Value, prop.OriginalValue) Then
                oldvals += String.Format("{0}{1}: {2}{3}",
                                         vbCrLf,
                                         prop.Name,
                                         prop.OriginalValue)
                newvals += String.Format("{0}{1}: {2}{3}",
                                         vbCrLf,
                                         prop.Name,
                                         prop.Value)
            End If
        End If
    Next
    change.OriginalValues = oldvals
    change.NewValues = newvals
End Sub

```

Blazing-Fast **GRID CONTROLS**

Optimized for .NET



Award-Winning Presentation Controls and Reporting Libraries



Learn more and download your **FREE** evaluation copy today
Visit [DEVEXPRESS.COM/GRIDS](http://DEVEEXPRESS.COM/GRIDS)
or Call Us (818) 844-3383

Devexpress™

PRESENTATION CONTROLS | REPORTING CONTROLS
BUSINESS APP FRAMEWORKS | IDE PRODUCTIVITY TOOLS

All trademarks and registered trademarks are the property of their respective owners.

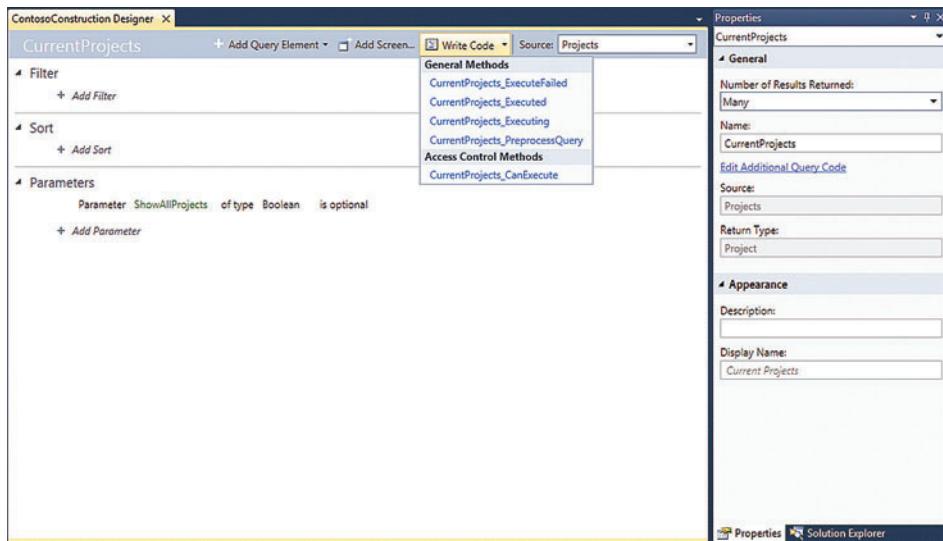


Figure 9 The CurrentProjects Query

classes for exporting construction project data to Word via COM. The bottom line is that you have a lot of flexibility to add your own custom code and classes to LightSwitch.

Working with the Save Pipeline

You can also add your own code to the server project in the same way. In the sample, there's an appointment screen that allows users to set up appointments between customers and employees. If the customer and the employee have e-mail addresses, an e-mail appointment (iCal) is sent to both parties when the appointment entity is inserted into the data source. The save pipeline exposes many methods that you can use to write business rules, call other services or kick off workflows, including Inserted/Inserting, Updated/Updating and Deleted/Deleting methods for every entity.

The helper class to send the appointment via SMTP e-mail is located in the Server project's \UserCode folder, and it's called SMTPMailHelper. The application also sends updates and cancellations when appointments are updated or deleted in the system. In order to access these save pipeline methods, select the entity in the designer and then the Write Code button drop-down. When you hover over the methods, the tooltip will tell you whether the code runs on the client, the server or both. In the case of the Appointment entity, we need to inject code into the server-side methods shown in **Figure 6** in order to track and send new, updated and canceled appointment e-mails.

You'll also work with the save pipeline when you implement an audit trail. Audit trails are common in business applications and are used to track what changes were made and by whom to records in the system. In the sample application, a table called EmployeeChange tracks changes to fields on the Employee, as shown in **Figure 7**.

To capture the new and original values on an Employee that's being updated, you can drill into the Details property of the entity. This allows you to get at the more advanced entity API. In the Employees Updating method, you can dynamically loop through all the storage properties on the entity and record their new and original values, as shown in **Figure 8**.

For more information on the save pipeline, validation framework and entity API, see the Working with Code section of the LightSwitch Developer Center (bit.ly/inJ3DE).

Advanced Queries

Because LightSwitch applications are all about data and screens, it's not surprising that you can create queries. The query designer provides a quick, simple way to define filters, sorts and query parameters, as well as specify whether the query should return one or many rows. You can also base queries on other queries, which comes in handy when you always have a filter or sort you want applied to a set of data. You can also specify required

and optional parameters. However, the query designer does have its limits. Luckily, you can click on the Write Code button at the top of the designer (or click "Edit Additional Query Code" in the properties window) and write code in the PreprocessQuery method.

In the Contoso Construction application, there's a query called CurrentProjects, as shown in **Figure 9**. The CurrentProjects query only defines an optional parameter through the designer; the rest of the query is handled in code.

Notice that this query only defines an optional parameter and doesn't specify a filter or sort. We need to check if the parameter is supplied and, if so, evaluate its value to determine if a filter should be applied. The PreprocessQuery method is passed any parameters that you define in the designer as well as the query itself. This means that you can define filter and sort clauses in the designer and then add to the query by writing LINQ code to filter or sort it further. Just keep in mind that you always need to return a collection of entity types that LightSwitch expects; you can't return projections or other types.

So, in the PreprocessQuery method, you can write code as in **Figure 10**.

Now when you create a screen based on the query, LightSwitch will see that there's an optional parameter, then automatically create that screen data item for you and bind it to the query parameter. Because

Figure 10 Intercepting Queries in the PreprocessQuery Method

```

Private Sub CurrentProjects_PreprocessQuery()
    ShowAllProjects As System.Nullable(Of Boolean),
    ByRef query As System.Linq.IQueryable(Of LightSwitchApplication.Project))

    'If ShowAllProjects is False (or Nothing) then just pull up the
    'projects that do not have an actual end date specified.
    If Not (ShowAllProjects.HasValue) Then ShowAllProjects = False

    If Not (ShowAllProjects) Then
        query = From p In query
                Where p.ActualEndDate Is Nothing
                Select p

    End If
End Sub

```

Rock-Solid **REPORTING CONTROLS**

Optimized for .NET



Award-Winning Presentation Controls and Reporting Libraries



Learn more and download your **FREE** evaluation copy today
Visit DEVEXPRESS.COM/REPORTING
or Call Us (818) 844-3383

Devexpress™

PRESENTATION CONTROLS | REPORTING CONTROLS
BUSINESS APP FRAMEWORKS | IDE PRODUCTIVITY TOOLS

All trademarks and registered trademarks are the property of their respective owners.

Show All Projects is a Boolean, it will create a checkbox, and when the user clicks the box the query will execute automatically.

I hope you're starting to see all the code hooks that LightSwitch provides; queries are no different. Another interesting query in the sample application uses an aggregate to find projects that are over budget. This query doesn't specify anything in the designer; it uses only code, because it needs to aggregate child data to calculate the materials cost on the project:

```
Private Sub ProjectsOverBudget_PreprocessQuery()
    ByRef query As System.Linq.IQueryable(Of LightSwitchApplication.Project)) 

    'Return projects where the cost is over the original estimate
    query = From p In query
        Let cost = p.Labor +
            (Aggregate m In p.ProjectMaterials Into Sum(m.Quantity * m.Price))
        Where cost > p.OriginalEstimate
        Order By p.StartDate
        Select p
End Sub
```

In the sample application, these two queries are presented on Search Data Screens so that users can export the results to Excel, analyze the data and create reports. Because these reports are used often, it makes sense to have them hardcoded. However, it would be nice to allow users to define their own custom reports. You can do just that by installing the Filter Control extension that's available in the Samples Gallery, which includes all the source code, at bit.ly/FyMEnK.

Using Extensions

LightSwitch extensions are installed like any other Visual Studio extensions. Click on the VSIX file to install, then restart Visual Studio. Open your LightSwitch Project Properties and you'll see an Extensions tab that allows you to select which extensions to use in the project, as shown in **Figure 11**.

To use the filter control extension, create a query and add a single string parameter called FilterTerm, then add the following code to the PreprocessQuery method:

```
query = LightSwitchFilter.Server.Filter(query, FilterTerm, Me.Application)
```

Next, add a screen based on the query and in the screen designer change the control type for the Filter Term control to "Advanced Filter Builder," and change the label position to "None." In the Contoso Construction sample application, the query is called CustomFilter-Projects and the screen is called CustomReport.

When you run the sample application and open the Custom Report screen, the filter builder control will be displayed. Once you've defined the filter, the Go button will execute it against the server. The filter is a Silverlight control that will generate an XML representation of the designed filter. This is then passed to the query as a string. On the server, the query code will parse the XML and generate appropriate filter clauses.

Another handy extension is the Excel Importer extension that you can get at bit.ly/e580r3, along with all the source code.

LightSwitch already has built-in support for exporting data in grids to Excel. The Excel Importer lets you *import* data from spreadsheets by mapping Excel columns to properties of an entity. This extension is used on the Materials Catalog screen in the sample application. Once you install and enable the extension, create an Editable Grid Screen and add a button to the screen command bar. In the Execute method for the command you only need to write one line of code:

```
Private Sub ImportFromExcel_Execute()
    LightSwitchUtilities.Client.ImportFromExcel(Me.Materials)
End Sub
```

As you can see, it's very easy to install and use extensions in LightSwitch to provide additional capabilities. When you package up your customizations as extensions, you can distribute them broadly to other LightSwitch developers.

Build Your Own Extensions

To build extensions, you'll need Visual Studio Professional SP1 or higher with LightSwitch installed, the Visual Studio SDK and the LightSwitch Extensibility Toolkit, which you can download from the LightSwitch Developer Center (msdn.com/lightswitch). This adds Visual Basic and C# extensibility project templates under the LightSwitch node in your New Project dialog.

Let's create a simple theme for our Contoso Construction application. First create a new LightSwitch Extension Library project; I'll call it ContosoThemes. This sets up all the necessary projects for the extension. You'll notice a similarity to the File View of a LightSwitch application, but there are some additional projects included, namely the LSPKG and the VSIX projects.

Next, right-click on the ContosoThemes.LsPkg and select Add New Item. Here you'll see the set of supported LightSwitch Extension templates. Select the LightSwitch Theme item and give it a name—

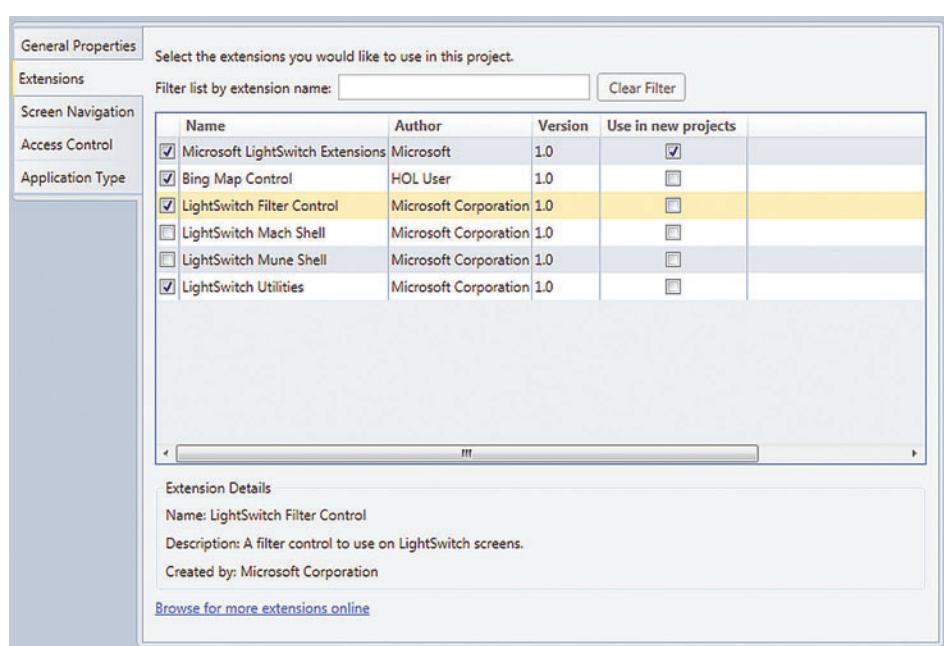


Figure 11 Enabling an Extension on the Extensions Tab of the LightSwitch Project Properties

Powerhouse **ANALYTICS**

Optimized for .NET



Award-Winning Presentation Controls and Reporting Libraries



Learn more and download your **FREE** evaluation copy today
Visit DEVEXPRESS.COM/ANALYTICS
or Call Us (818) 844-3383

DevexpressTM

PRESENTATION CONTROLS | REPORTING CONTROLS
BUSINESS APP FRAMEWORKS | IDE PRODUCTIVITY TOOLS

All trademarks and registered trademarks are the property of their respective owners.

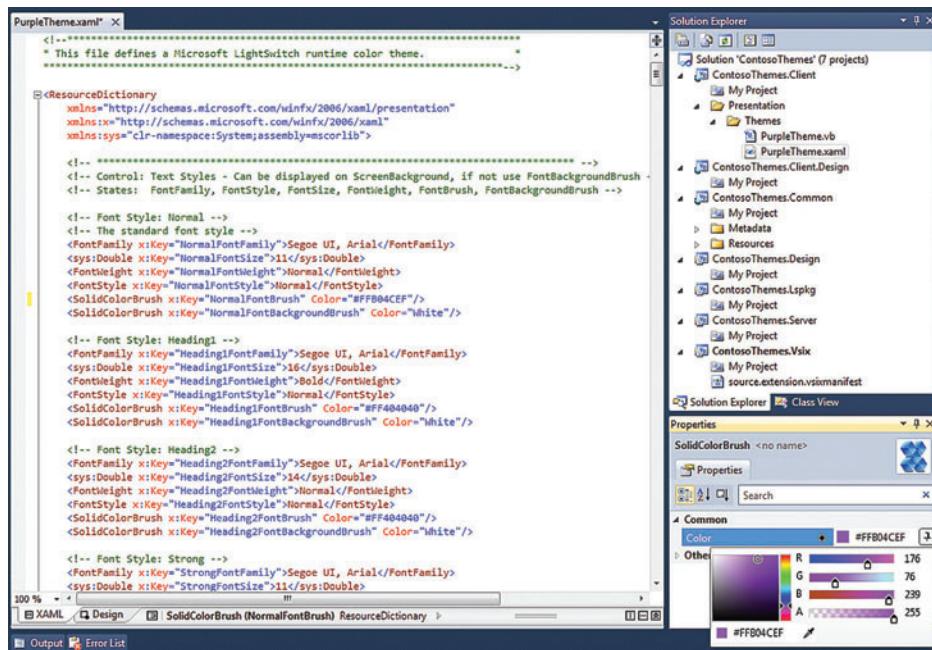


Figure 12 Creating a LightSwitch Theme Extension

I'll call it PurpleTheme. The new theme's pre-canned XAML file will be created and opened in the IDE. By default, it has all the brushes and styles that are in the standard shell theme. You can use any tool you like to set the colors and styles for every item that LightSwitch displays in its shell. For this example, I'll just change a few colors using the XAML editor and the properties window to select colors, as shown in **Figure 12**.

Once you're ready to test your extension, you can hit F5 to launch a debug IDE instance. To test the theme, create a new LightSwitch project in this instance. Open the LightSwitch Project Properties

and on the Extensions tab enable the ContosoThemes extension you just built. Next, switch to the General tab and select the PurpleTheme. You'll need to create entities and screens so you can see your theme in action; then you can press F5 in your test project to see what it looks like.

When you're happy with your theme, build the extension in Release mode—then you can hand the VSIX package located in the \bin\Release folder of the VSIX project to a LightSwitch developer to use on any of her projects. You can also upload it to the Visual Studio Gallery for broad distribution. **Figure 13** shows what the theme looks like in the Contoso Construction application.

For more information on building LightSwitch Extensions, see the Advanced Topics section of the

LightSwitch Developer Center (bit.ly/h0MP7P).

Wrapping Up

LightSwitch dramatically simplifies the development of data-centric business applications because it takes care of all the plumbing for you. It allows you to concentrate on the business logic and other custom features that your users need to get their jobs done efficiently. There are many levels of customization available to LightSwitch developers, as well as to professional developers looking to build LightSwitch extensions for the community. I hope this article has shown you how flexible and customizable LightSwitch can be. For information, training, samples, videos, forums, community and more, please visit the LightSwitch Developer Center at msdn.com/lightswitch.

Enjoy!

BETH MASSI is a senior program manager on the Microsoft Visual Studio BizApps team, which builds the Visual Studio tools for Windows Azure, Office and SharePoint, as well as Visual Studio LightSwitch. Massi is a community champion for business application developers and is responsible for producing and managing online content and community interaction for the BizApps team. She has more than 15 years of industry experience building business applications and is a frequent speaker at software development events. You can find her on a variety of developer sites, including MSDN Developer Centers, Channel 9 and her blog, BethMassi.com. Follow her on Twitter at twitter.com/BethMassi.

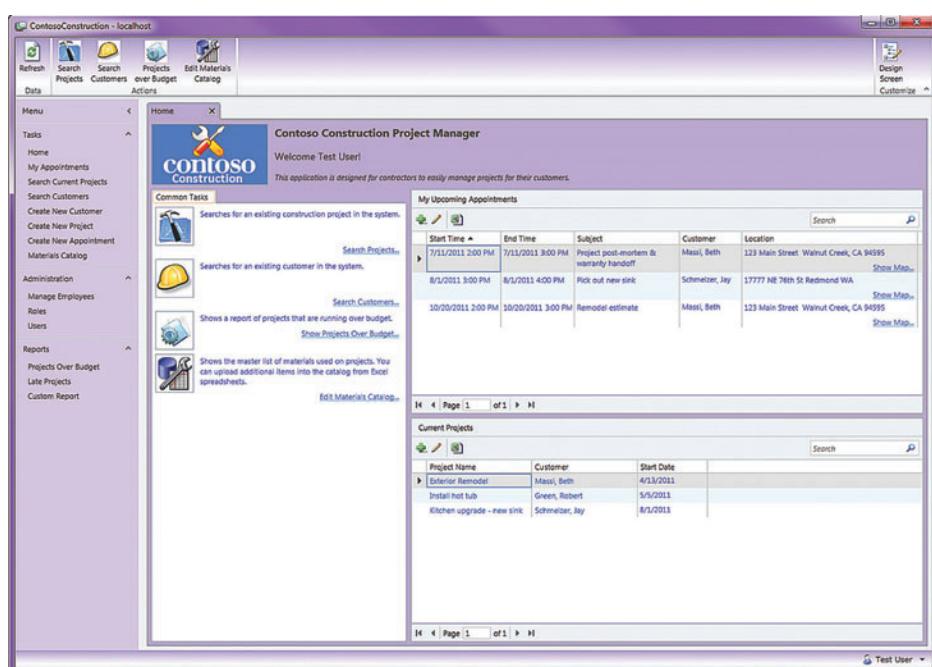


Figure 13 Applying a Theme Extension to a LightSwitch Application

Elegant **CHARTING CONTROLS**

Optimized for .NET



Award-Winning Presentation Controls and Reporting Libraries



Learn more and download your **FREE** evaluation copy today
Visit DEVEXPRESS.COM/CHARTING
or Call Us (818) 844-3383

Devexpress™

PRESENTATION CONTROLS | REPORTING CONTROLS
BUSINESS APP FRAMEWORKS | IDE PRODUCTIVITY TOOLS

All trademarks and registered trademarks are the property of their respective owners.

Build Business Applications with Visual Studio LightSwitch

Robert Green

Consider the tale of two overworked individuals. Antonio is a senior developer in the IT department of a large bank with hundreds of branch offices. Dalia is the manager at one of these branches. She wants to do a better job of tracking computer assets. Who has what computer and how long have they had it? When was each printer and fax machine purchased and serviced? How much does the branch spend on peripherals and supplies each month?

Dalia e-mails Antonio and asks him to create an asset-tracking application. Antonio agrees this is needed and could save the bank

significant sums of money, but he's already fully committed to other projects and has no time to help Dalia. He knows that Dalia is going to build the application herself in Excel or another end-user application, and this worries him because she's likely going to build a single-tier, self-contained application that won't scale. But what can he do?

Fast forward six months. Dalia is a hero. She built her app and it has cut her branch's computer spending by 25 percent while also increasing productivity due to more efficient use of resources. Bank management decides that all of the branch offices should adopt this system and that the data should be centralized. Antonio is told to make this happen by the end of the month, and it's now his nightmare to convert this application into something that resembles what he would have built if he had had the time. Good luck, Antonio!

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

This article discusses:

- Defining data with entities
- Custom validation rules
- Creating and using queries
- Controlling what actions users can take
- The three LightSwitch deployment models

Technologies discussed:

Visual Studio LightSwitch, Silverlight, Windows Azure, Model-View-ViewModel, Microsoft .NET Framework

Code download available at:

code.msdn.microsoft.com/Contoso-Construction-9f944948

Introducing Visual Studio LightSwitch

Visual Studio LightSwitch is designed to address this all-too-common scenario. LightSwitch is a new product in the Visual Studio family aimed at developers of all skill levels who want to quickly create data-centric business applications for the desktop, Web and cloud. LightSwitch simplifies the development process because it does most of the development work for you. You don't need to write code to interact with databases and you don't need to manually lay out screens. You can concentrate on the business logic.

LightSwitch applications are based on Silverlight. They use proven n-tier architecture patterns based on Model-View-ViewModel

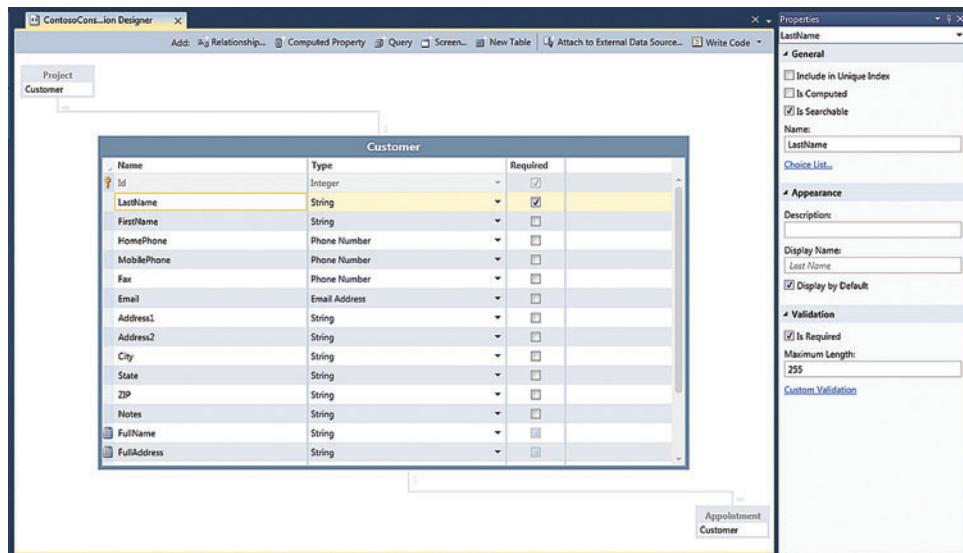


Figure 1 The Customer Entity

(MVVM), as well as familiar Microsoft .NET Framework technologies such as the Entity Framework and WCF RIA Services. LightSwitch applications can be deployed as desktop (out-of-browser) or browser-based applications. Desktop applications can leverage local hardware resources and work with applications such as Microsoft Word and Excel. Both desktop and browser-based LightSwitch applications can be hosted on IIS and Windows Azure.

The primary audience for LightSwitch is end-user developers such as Dalia. They're information workers, IT professionals, business analysts and so on who do some development as part of their job. They aren't professional developers, and they want a smooth on-ramp to development, built-in plumbing that handles common application requirements, and simple and flexible deployment options. In short, they want to quickly build well-architected, data-centric applications that can be easily deployed and scaled.

Existing Visual Studio developers will often find LightSwitch an attractive addition to their toolset. LightSwitch installs on top of Visual Studio 2010 Professional and above. Visual Studio developers can build LightSwitch applications from scratch or they can open a LightSwitch application in Visual Studio and extend it. If Dalia had used LightSwitch to build her application, Antonio could open it and enhance it. Or Antonio could build the application in LightSwitch in potentially dramatically less time than if he started from scratch.

In this article, I'll review a LightSwitch application used by the construction company arm of the ubiquitous Contoso Corp. The company wants to manage customers and their projects. You can download the sample application I'll be discussing here: code.msdn.microsoft.com/Contoso-Construction-9f944948.

Entities

The first step in building a LightSwitch application, after choosing whether you want to code in Visual Basic or Visual C#, is to define your data. You can create new tables or attach to external data sources. If you create tables from scratch, they're added to the application database, which is a SQL Server Express database. Note that when it's time to deploy the application, you can choose any edition of

SQL Server to host the data. To work with external data, you can connect to an external database such as SQL Server or SQL Azure or any database for which you have an Entity Framework provider. You can also connect to SharePoint lists or WCF RIA Services.

The sample application contains a Customer entity (table), as shown in **Figure 1**.

For each entity property, you can define not only the data type, but also whether the property is required, unique, searchable and displayed on screens by default. You can also specify alternate display labels and maximum lengths. For numeric and date

properties, you can specify minimum and maximum values.

Rather than leave certain properties as strings or their default types, you can use built-in data types, also referred to as *custom business types*. The phone and fax properties in the Customer entity use the Phone Number data type, and the Email property uses the Email Address data type. The built-in data types provide validation and will generate user-friendly runtime errors. There's no need for you to write code to validate that a phone number or e-mail address is valid. In addition, the Phone Number data type includes built-in formatting, as shown in **Figure 2**.

There are also Date, Image and Money data types that all come with built-in editors, formatting and validation.

After adding properties to an entity, you should review the entity's Summary property, which describes the entity. It also becomes a hyperlink on search screens.

The screen displays all customers in a grid. Notice the Full Name column contains hyperlinks. When the user clicks a full name, LightSwitch displays the detail screen for that customer. As the LightSwitch developer, you don't need to write any code to make this happen. **Figure 3** shows the Search Customers screen.

Figure 2 The Built-In Phone Number Data Type Provides Validation and Formatting

Full Name	Last Name	First Name
▶ Green, Robert	Green	Robert
Schmelzer, Jay	Schmelzer	Jay

Figure 3 An Entity's Summary Property Is a Hyperlink in Search Screens

By default, LightSwitch uses the first string property as the entity's Summary property. If there are no strings, it will use the first non-string property. In the sample application, customers are people and the first string in the Customer entity is LastName. Rather than use that as the Summary property, the Customer entity has a FullName property. This is a computed property. To make a property computed, you check *Is Computed* in the Properties window or click a button at the top of the designer, click Edit Method, and then write the following code to compute the value of the property (I'll be using Visual Basic):

```
Private Sub FullName_Compute(ByRef result As String)
    result = Me.LastName + ", " + Me.FirstName
End Sub
```

To make FullName the Summary property, simply select the Customer entity and set the Summary property to FullName.

Computed properties aren't stored in a database. They're computed at run time as properties of an entity. You can use computed properties for simple concatenations of names and addresses and also for calculated values such as year-to-date revenue or total value of outstanding invoices.

To create a relationship between entities, click the Relationship button in the Data Designer. In the Add New Relationship dialog box, select the two entities, the type of relationship and the delete behavior. The Customer entity has one-to-many relationships

Figure 4 The Address1_Validate Method

```
Private Sub Address1_Validate(results As EntityValidationResultsBuilder)
    'Warn the user if the Address is empty
    If Me.Address1 = "" Then
        results.AddPropertyResult(
            "Address should not be empty. " &
            "Construction project cannot begin unless an address is supplied.", 
            ValidationSeverity.Warning)
    End If
End Sub
```

Figure 5 The ZIP_Validate Method

```
Private Sub ZIP_Validate(results As EntityValidationResultsBuilder)
    If Me.ZIP <> "" Then
        'Enter the dash if not supplied and is 9 digits long
        If Me.ZIP.Length = 9 Then
            Me.ZIP = Me.ZIP.Substring(0, 5) + "-" + Me.ZIP.Substring(5)
        End If
        'Make sure valid zip code (5 or 5+4 format)
        If Not System.Text.RegularExpressions.Regex.IsMatch(
            Me.ZIP, "^\\d{5}$|^\\d{5}-\\d{4}$") Then
            results.AddPropertyError(
                "Please enter a valid US ZIP code. (ex. 98052 or 98052-1234)")
        End If
    End If
End Sub
```

defined with the Project and Appointment entities. Therefore, Customer has a Projects property and an Appointments property. These two properties are collections.

You can not only create relationships between entities in a single data source, but also between entities in multiple data sources. LightSwitch will handle retrieving all of the data, presenting it to users and saving changes. The ability to create federated relationships is a unique and compelling feature of LightSwitch.

Custom Validation Rules

In addition to using the validations provided by the built-in business types, you can add custom business logic code, both at the screen and entity level. Screen validation code runs only on the client and validates screen properties and data. Entity property validation code runs on the client first and then on the middle tier. Users get immediate feedback and can correct errors before sending data to the middle tier. The validation logic also runs on the middle tier to handle situations where data is changed by other users. This is a best practice in *n*-tier design.

You can not only create
relationships between entities
in a single data source,
but also between entities in
multiple data sources.

To write validation code for an entity property, you can select the property in the Entity Designer and then select the appropriate method from the Write Code button drop-down list: Address1_Validate, for example (shown in Figure 4).

EntityValidationResultsBuilder is a container for validation results. It can contain validation information, warnings and errors. Validation information and warnings present information to the user but don't prevent the user from saving data. If the collection contains any validation errors, the user can't save the data.

In the sample application, the user is warned if the address is empty. The user can still save the data. However, the code to validate the ZIP code (shown in Figure 5) isn't as forgiving.

If the ZIP code isn't in the proper format, the code adds a validation error. This will prevent the user from saving the invalid data. Figure 6 shows the results of leaving the address empty and entering an invalid ZIP code. The empty address generates a warning and the invalid ZIP code generates an error.

Screens

After you define your entities, the next step in building a LightSwitch application is to design screens. LightSwitch includes the following predefined screen templates:

- **Details Screen** This displays a single entity and can include related data in a grid.

The screenshot shows a LightSwitch screen titled "Search Customers" with a query result for "Green, Robert - Cust...". A yellow bar at the top indicates "2 validation issues exist with the data on this screen. Click here for more details." Below this, two error messages are displayed: "ZIP (Green, Robert): Please enter a valid US ZIP code. (ex. 98052 or 98052-1234)" and "Address1 (Green, Robert): Address should not be empty. Construction project cannot begin unless an address is supplied." The screen contains fields for Last Name, First Name, Home Phone, Mobile Phone, Address, City, State, and ZIP. The ZIP field is highlighted in red.

Figure 6 The Empty Address Generates a Warning While the Invalid ZIP Code Generates an Error

The screenshot shows the LightSwitch Screen Designer for the "SearchCustomers" screen. On the left is a tree view of the screen members, including sections like "Customers" and "Selected Item" with various properties like Id, LastName, FirstName, etc. In the center, there's a "Rows Layout" section containing a "Screen Command Bar" and a "Data Grid" section for the "Customers" entity. The "Properties" pane on the right shows settings for the "ScreenLayout" control, such as "Name: ScreenLayout", "Display Name: Search Customers", and "Control Type: Default (Rows Layout)".

Figure 7 The Screen Designer Shows a Hierarchical View of the Controls on a Screen

- Editable Grid Screen** This enables the editing of one or more items in a grid.
- List and Details Screen** This displays a collection of items in a list. Selecting an item in the list displays the details for that item.
- New Data Screen** This is a screen for creating a new item. The screen can also provide a grid to enable the ability to add related data at the same time.
- Search Data Screen** This displays data returned from a query. Each item includes a link to display the entity's details screen.

These templates provide a good starting point for your screen layouts. To create a screen, you can right-click the solution or the Screens node in the Solution Explorer and select "Add Screen." You can also click the Screen button in the Data Designer. When you create a screen, you select the type of screen and the primary screen data. You can select an entity or a query based on an entity. You can then specify related data you want to appear on the screen.

When you open a Silverlight screen in Visual Studio, you see a design canvas and XAML. When you open a screen in LightSwitch, you see the Screen Designer, as shown in Figure 7.

On the left is the screen members list. This contains the items that are available on the screen. It includes the data items in each entity included in the screen. It also contains methods such as Close, Refresh and Save. On the right is the screen content tree. This is a hierarchical view of the controls on the screen and the data to which they're bound. You can add data items to the screen as needed. You can rearrange the screen controls as well as modify the screen layout. For example, you may want the customers list on the left and orders on the right rather than customers on top and orders below. You can also set various properties of controls, such as label text, whether a label appears, horizontal and vertical alignment, height and width.

After you create at least one screen, you can press F5 and run the application. The first screen you create is the application's default screen. You can change this on the Navigation tab of the Application Designer. LightSwitch applications automatically include an application shell, a navigation menu, a ribbon, a tabbed area for screens and data binding, as shown in Figure 8.

The application also has built-in dirty checking and concurrency handling. LightSwitch handles all the plumbing of a typical data-centric application for you so you don't need to write any of that code.

The LightSwitch screen templates provide specific functionality.

The LightSwitch screen templates provide specific functionality. For example, a search screen contains a grid to display items and a button to export those items to Excel. You can add additional items as needed. The SearchCustomers screen (shown in Figure 8) has two additional buttons: an Add Customer button in the ribbon and an Add button in the grid's header.

To add a button to the ribbon, you can right-click Screen Command Bar in the Screen Designer and select Add Button. Or you can expand the Screen Command Bar and select New Button from the Add button's drop-down list, as shown in Figure 9. To add a button to the grid, use the Command Bar contained in the grid.

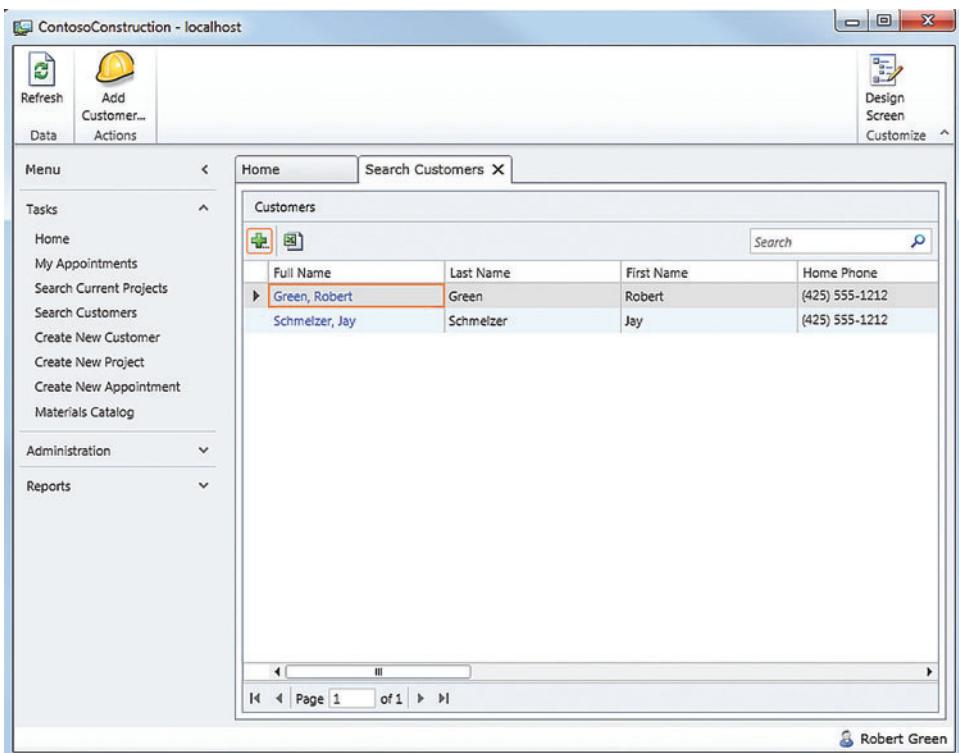


Figure 8 LightSwitch Applications Automatically Include an Application Shell, Menu and Ribbon

When you add a button, LightSwitch prompts you for the name of the method that runs when the user clicks the button. You can right-click the button and select Edit Execute Code, and then write the code that executes. Both Add buttons on the SearchCustomers screen call the gridAddAndEditNew_Execute method, which calls the ShowCreateNewCustomer method. This is a LightSwitch built-in method that displays the CreateNewCustomer screen, as shown here:

```
Private Sub gridAddAndEditNew_Execute()
    Me.Application.ShowCreateNewCustomer()
End Sub
```

Just like entities, screens have a number of events you can handle, as shown in **Figure 10**.

The Run event occurs when a request is made to display a screen. The Run event handler includes a *handled* argument. If you set this to true, you can prevent the screen from continuing. The InitializeDataWorkspace event occurs just before the screen data

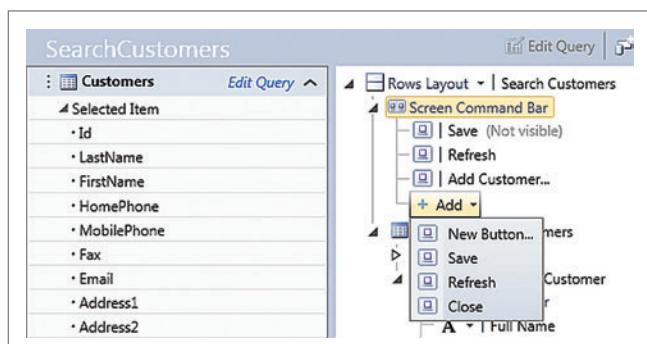


Figure 9 The Screen Command Bar Contains the Buttons in the Application's Ribbon

is retrieved. This is a good place for screen initialization code. The other events are self-explanatory. You might think that events such as Run or Saving are good places to put code that checks whether the user can perform actions such as open the screen or save the data. However, you should put that code in the appropriate Access Control methods, such as CanRun at the screen level or CanUpdate at the entity level. Access control is a big feature of LightSwitch applications that I'll discuss later.

LightSwitch development is designed to be highly iterative. Developers can edit screens at run time while running in Debug Mode by clicking the Design Screen button in the ribbon. This switches to the screen's Customization Mode, as shown in **Figure 11**.

Users can rearrange controls and set properties, click Save and see their changes immediately.

Knowing that a LightSwitch application is a Silverlight application, you may be wondering, "Where is the XAML?" A primary goal of LightSwitch is to make it much easier to build applications. It therefore does not expose XAML at design time. Rather, LightSwitch generates XAML at run time based on the screen design. This makes it dramatically easier for users to build applications. If you're an experienced Silverlight developer, you might be thinking that this also limits your ability to build the screens you'd like.

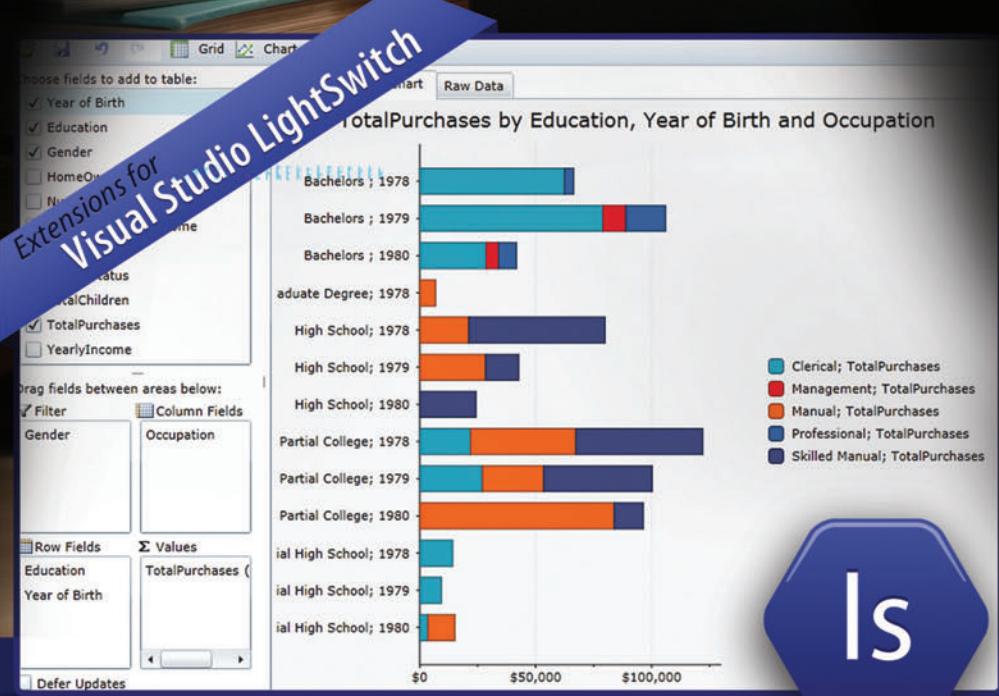
LightSwitch development is designed to be highly iterative.

While this may be true if you're only using Visual Studio LightSwitch, you have much more flexibility if you have LightSwitch on top of Visual Studio 2010 Professional or above. You can create your own Silverlight user controls and add them to LightSwitch screens, as well as use controls that aren't included in LightSwitch. You can create composite controls or even entire screens that include custom logic, plus you can easily bind these user controls to screen entities.

Another way to add functionality to a LightSwitch application is to use an extension. There are six types of LightSwitch extensions: controls, screen templates, business types, shells (application look and feel), themes (colors and brushes for a shell) and custom data sources. You'll need Visual Studio 2010 Professional or above and the Visual Studio 2010 SDK to create a LightSwitch extension, but anyone can use them in their LightSwitch project, no matter what edition they have installed.

BUSINESS INTELLIGENCE

in the Flip of a Switch



componentone.com/lightswitch

 Component**One**

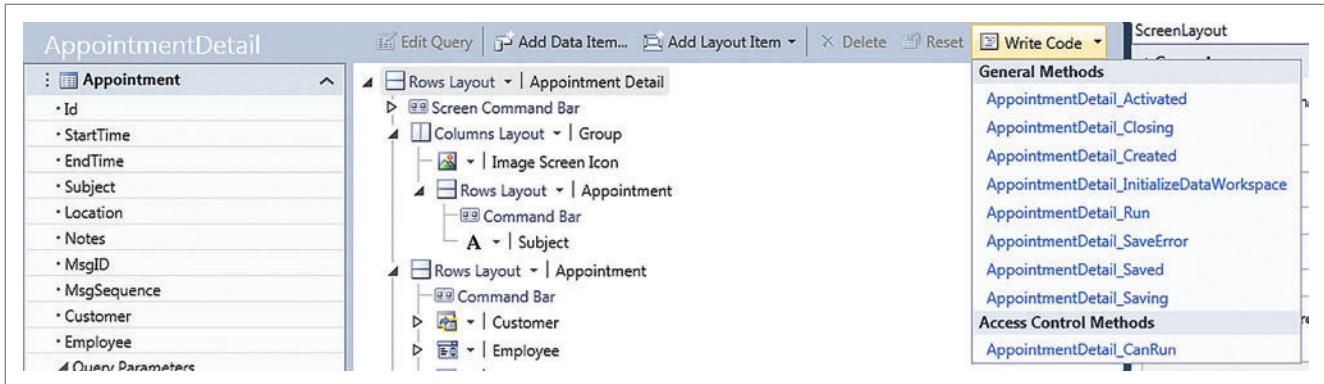


Figure 10 Access Screen Event Handlers via the Write Code Button

Extensions are distributed via VSIX packages. If you want to make an extension broadly available, you can upload it to the Visual Studio Gallery. It will then appear in the Extension Manager within LightSwitch. After installing an extension, you can enable it on the Extensions tab of the Application Designer. The Contoso Construction sample application uses a Bing map control extension to show the location of customers, as shown in **Figure 12**.

Queries

All screens are based on queries. The query for a Detail screen returns one row. The query for a List and Details screen returns one parent row and all of the related child rows. The query for a Search or Editable Grid screen will return all rows by default. Note that queries that return a number of rows don't literally return all of the rows at once. The queries support paging by default. To configure the paging, select the query in either the Screen or Query Designer and change the paging-related properties.

You can filter the data returned by a screen query that returns a collection of entities. To do this, click Edit Query in the Screen Designer. In the Query Designer, you can then add Where and Sort clauses. You can also add parameters to queries.

Screen queries are specific to a particular screen. Although it's quick and easy to modify a screen's query, a better practice is to create a reusable query. For example, you may want customers sorted by company name on the customer search screen. However, customers may also appear in a modal window picker on the screen where you create new appointments and in an autocomplete text box on the screen that displays customers and their projects. Rather than add the same Sort to three queries on three different screens, you can create one query and use it on the three screens.

To create a query, you can right-click on an entity in the Solution

Designer and select Add Query. You can name the query and then add filtering, sorting and parameters. The CurrentAppointments query, shown in **Figure 13**, returns all appointments, sorted by start time, where the start time is now or later.

LightSwitch uses the standard ASP.NET membership and role providers to enable both Windows and Forms authentication.

Once you've created a query, you can use it as the basis for screens. You can also use it as the basis for additional queries. For example, in the sample application, the CurrentAppointmentsByEmployee query starts with the CurrentAppointments query and then filters the results to return only the appointments for a particular employee.

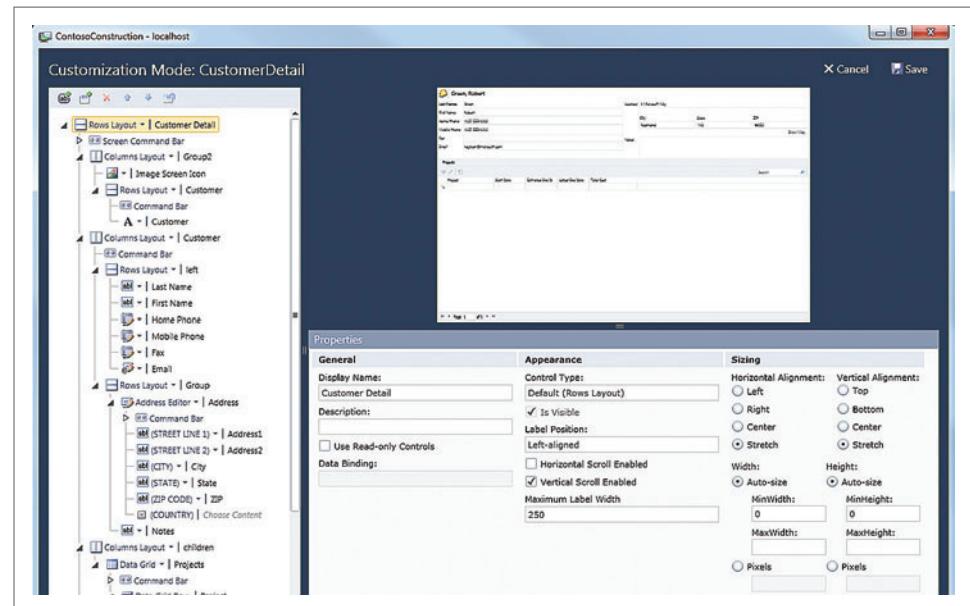


Figure 11 Users Can Edit Screens at Run Time and View Their Changes Immediately

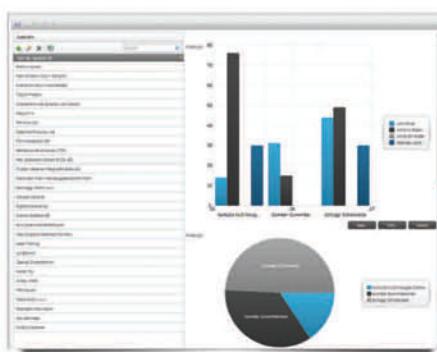


NetAdvantage® for Visual Studio® LightSwitch™

Code-Free Dashboards and Data Visualizations

RICH DASHBOARD EXPERIENCES

Build informative data visualizations to empower your decision makers.



PRESENT ENGAGING INTELLIGENCE

Enlighten your users with Digital Gauges that can display 7-segment numeric or 14-segment alphanumeric readouts.



SCAN HERE
for an exclusive
look at LightSwitch!
www.infragistics.com/ls

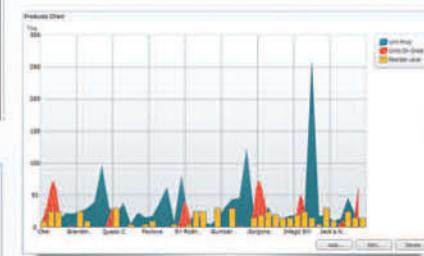


MODERN LOOK AND FEEL

Fast switching between screens with the visually appealing OutlookBar and TileView Shell.

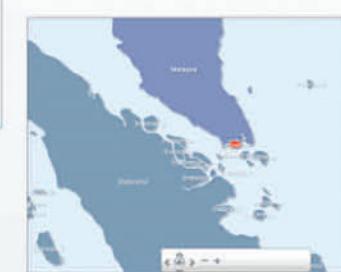
CUSTOM THEMES

Develop "in" style by applying our IG theme or Office 2010 Blue theme to your applications.



MULTIPLE CHARTS

Add chart layers and indicators to our popular Area, Column, Line, Spline, Step, Range, Waterfall or even Pie charts.



INTERACTIVE MAP CONTROL

Visualize anything from floor plans to warehouse content with features like panning, zooming, image tiling, mouse overs and more.

TAKE YOUR WEB APPLICATIONS TO THE NEXT LEVEL WITH OUR TOOLS
INFRAGISTICS.COM/LIGHTSWITCH



INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

Access Control

Access control gives you the ability to control what actions users can take in an application. LightSwitch uses the standard ASP.NET membership and role providers to enable both Windows and Forms authentication. You can authorize users to perform actions by creating permissions, assigning them to users and then checking in code whether the user has a particular permission.

By default, all users can perform all actions in a LightSwitch application. To change this, open the Application Designer and select the Access Control tab, as shown in **Figure 14**. Then select either Windows or Forms authentication. The built-in SecurityAdministration permission controls whether a user can see the security administration screens at run time. You use those screens to assign permissions to users and add users if necessary. You can create additional permissions if desired. During development, you can turn off permissions by unchecking Granted for debug. This enables you to test the application using various combinations of permissions.

You can check for permissions in code at the entity, screen and query levels. Entities provide CanDelete, CanInsert, CanRead and CanUpdate methods, which all run on the server. You can access these from the Write Code button drop-down list in the Data Designer. You can check for the appropriate permission and return false if the user isn't permitted to take the associated action. For example, in the sample application, only administrators can modify employee data. The following code ensures this:

```
Private Sub Employees_CanDelete(ByRef result As Boolean)
    result = Me.Application.User.HasPermission(
        Permissions.SecurityAdministration)
End Sub
Private Sub Employees_CanInsert(ByRef result As Boolean)
    result = Me.Application.User.HasPermission(
        Permissions.SecurityAdministration)
End Sub
Private Sub Employees_CanUpdate(ByRef result As Boolean)
    result = Me.Application.User.HasPermission(
        Permissions.SecurityAdministration)
End Sub
```

Screens provide a CanRun method, which runs on the client. You can use this method to not open a screen if the user can't view or modify the screen's data. The following code ensures that only administrators can open the ManageEmployees screen:

```
Private Sub ManageEmployees_CanRun(ByRef result As Boolean)
    result = Me.User.HasPermission(Permissions.SecurityAdministration)
End Sub
```

During the deployment process, you'll specify the user name and password of the administrator. At run time, the administrator creates roles and assigns users to them. The administrator then assigns permissions to roles.

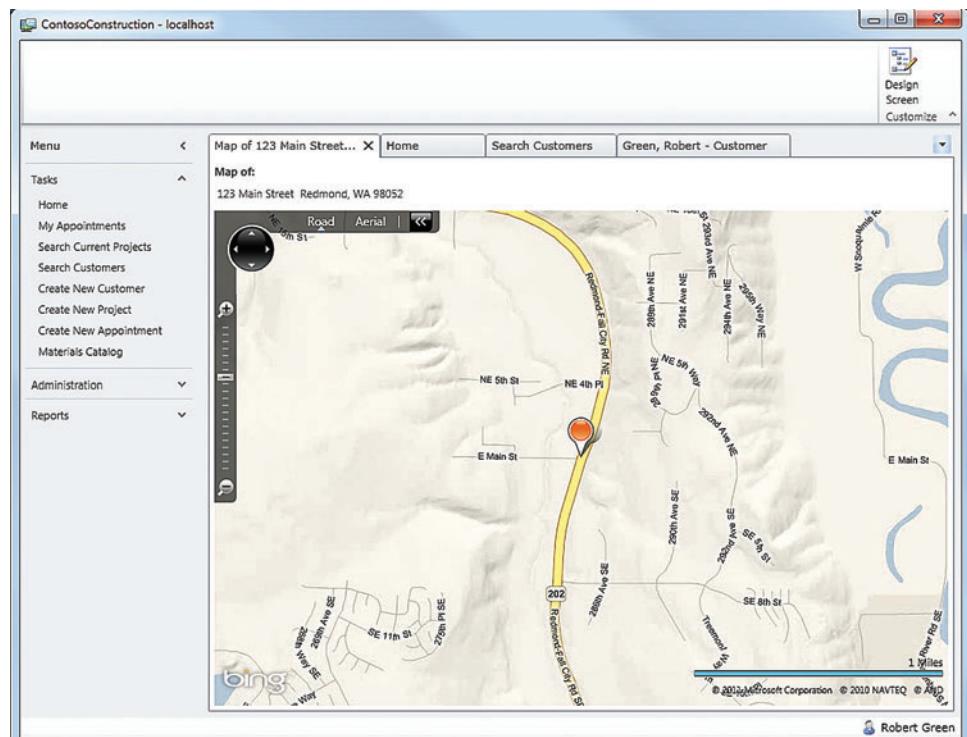


Figure 12 You Can Use LightSwitch Extensions, Such as the Bing Map Control Extension, to Provide Additional Functionality in Your Applications

Deployment

LightSwitch provides three models for deploying applications: two-tier desktop application, three-tier desktop application and three-tier Web application.

A two-tier desktop application runs entirely on the end user's computer as an out-of-browser Silverlight application. The UI and all the application's middle-tier components run locally. The application connects to the database directly in typical client-server fashion. This avoids the need for a Web server. The application has access to local resources, including COM or local files. COM support provides the ability to control applications such as Word or Excel. Note that desktop applications require Windows.

A three-tier desktop application runs as an out-of-browser Silverlight application hosted on IIS or Windows Azure. The UI runs on the end user's computer, while the middle-tier components run on the host server.

A three-tier Web application runs as an in-browser Silverlight application hosted on IIS or Windows Azure. The UI is browser-based and the middle-tier components run on the host server. Web applications don't have access to COM or local resources, but they do provide the most reach across Mac and Windows OSes and multiple browsers.

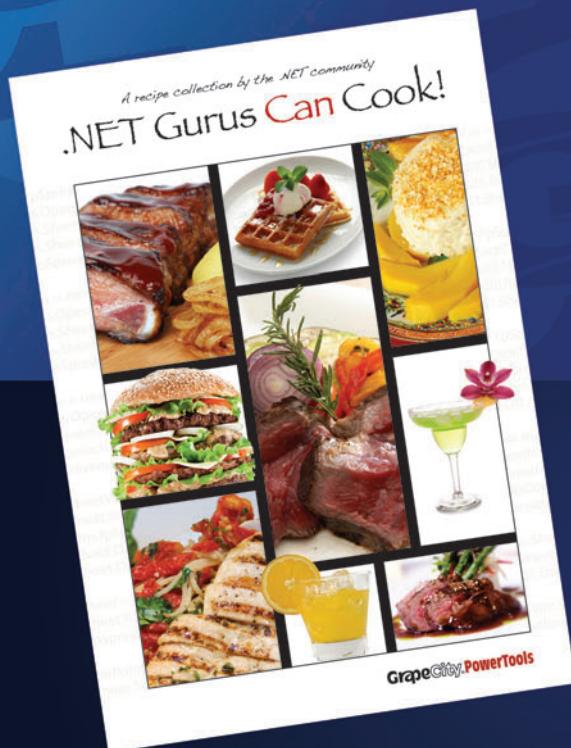
To deploy an application, you first publish it. To publish the application, click the Publish button on the Application Type tab in the Application Designer, as shown in **Figure 15**.

The LightSwitch Publish Application Wizard will then walk you through the publishing process.

If you publish the application as a two-tier desktop application, you'll create a ClickOnce package. You'll create a SQL Server



Real Recipes by Real .NET Influencers!



*Download your
FREE cookbook
and enjoy a great
meal tonight!*

www.GCPowerTools.com/cookbook



brought to you by:



 **GrapeCity PowerTools**

www.GCPowerTools.com



Watch the Hottest News and see the Freshest Development through the eyes of Russ Cam!

GvTv.GCPowerTools.com



brought to you by:



Watch Russ Fustino as he travels across the country visiting .NET UserGroups, CodeCamps and Conferences talking to the leaders who are paving the way in our industry.

GrapeCity PowerTools
www.GCPowerTools.com

The screenshot shows the LightSwitch query editor interface. At the top, there's a toolbar with buttons for 'Add Query Element', 'Add Screen...', 'Write Code', and a dropdown for 'Source' set to 'Appointments'. Below the toolbar, there are three main sections: 'Filter', 'Sort', and 'Parameters'. The 'Filter' section contains a 'Where' clause with 'StartTime >= Now'. The 'Sort' section has 'Sort by StartTime' and 'Ascending' selected. The 'Parameters' section is currently empty.

Figure 13 You Can Filter the Results of a Query and Specify the Sort Order

database that contains any local tables you created, as well as system tables. You can also specify where this database resides. If the application is used by a single person, you may locate the database on that user's computer and use SQL Server Express. If the application will be used by more than one person, you'll likely store the database on a network computer running SQL Server.

If you publish the application as a three-tier desktop or Web application hosted on IIS, you can publish directly to IIS if the server is running the Microsoft Web Deployment Tool service. Otherwise, you can create an MSDeploy package and manually import it into IIS. If you publish the application to Windows Azure, the wizard will prompt you for your account subscription ID, the service and storage accounts you'll use, and an SSL certificate to use. For more information on deployment and publishing to Windows Azure, see the deployment section of the LightSwitch Developer Center on MSDN (bit.ly/jiYov5).

The screenshot shows the 'Application Type' tab of the LightSwitch application properties dialog. It includes sections for 'Access Control' and 'Application Type'. Under 'Access Control', it says 'Select the type of authentication to use:' with 'Use Windows authentication' selected. It also lists 'Select who is allowed to use your application:' with 'Allow any authenticated Windows user' selected. Under 'Application Type', it says 'Define permissions or select permissions to use for debugging:' and shows a table:

Name	Display Name	Description	Granted for debug
SecurityAdministration	Security Administration	Provides the ability to manage security for the application.	<input checked="" type="checkbox"/>
<Add New Permission>			<input type="checkbox"/>

Figure 14 Specify Windows or Forms Authentication and Then Specify Additional Permissions

The screenshot shows the 'Application Type' tab of the LightSwitch application properties dialog. It includes sections for 'Client' and 'Application Server'. Under 'Client', it says 'Choose the client and server topology for your application:' with 'Desktop' selected. It says 'This application runs on the user's desktop and can access other programs on the user's computer.' Under 'Application Server', it says 'Run application services on the end user's machine.' It also says 'Host application services on an Internet Information Services (IIS) Server. A separate server is required.' and 'Host application services using Windows Azure.' There is a 'Learn more about hosting using Windows Azure' link. At the bottom right is a 'Publish...' button.

Figure 15 Specify Whether the Application Is a Two-Tier Desktop Application, Three-Tier Desktop Application or Three-Tier Web Application

The Simplest Way

Wrapping up, the primary audience for LightSwitch is end-user developers. These are people building applications to support business functions. They aren't professional developers. They're IT pros, information workers and others who do some development as part of their job. They often have a need to build applications that manage "things," such as department com-

puter assets or car fleets. They may need an application to manage an event, such as a quarterly open house.

LightSwitch is the
simplest way to build data
applications for the desktop
and the cloud.

LightSwitch provides these users with a smooth on-ramp to development. It automatically builds the "plumbing" to perform common application tasks, such as working with data, generating screens, exporting data to Excel and more. It also provides a simple and flexible deployment model.

LightSwitch is the simplest way to build data applications for the desktop and the cloud. The Dalias of the world can build the applications they need and then turn them over to the Antonios of the world to be extended and deployed. For more information on how to build applications with LightSwitch, visit the LightSwitch Developer Center on MSDN (msdn.com/lightswitch). ■

ROBERT GREEN is a technical evangelist in the Developer Platform and Evangelism group at Microsoft. This is his second stint at Microsoft. From 2005 to 2010, he was a senior consultant with MCW Technologies, focused on developer training. He authored and coauthored a number of Visual Studio and .NET courses for AppDev (appdev.com). Prior to that, in his first stint at Microsoft, he was in Developer Tools marketing and then community lead on the Visual Basic team.

THANKS to the following technical expert for reviewing this article: Beth Massi

Experience the Devexpress Difference

“ As a training, mentoring and consulting company, we are often put on the spot as to which vendors we like for .Net tools. There is only one answer from my company and that is Developer Express. We have used Developer Express tools in our projects for the past five years and the company continues to impress me with the quality of their tools.

They simply work. You get what you pay for. Mileage will vary with other vendors but I can assure you Developer Express is a sure bet. „

–Mark Dunn, MCT, MCAD, MCDBA, MCSD .Net

“ Our flagship product needed extensive visualizations including charts and graphs. We were looking for a single charting system that could address all of these needs, and handle large volumes of data. It needed to look attractive yet blend into our application.

With *XtraCharts* we were able to create high performance, real-time graphs of performance data through to detailed analytical bar charts. *XtraCharts* was the only option that was fast enough to handle the tens of thousands of data points our customers routinely throw at it. „

–Kendall Miller

Read More User Comments at:
DevExpress.com/Comments





Award-Winning Presentation Controls and Reporting Libraries

For a **FREE** trial version visit us at DevExpress.com/FreeEval

PRESENTATION CONTROLS | REPORTING CONTROLS
BUSINESS APP FRAMEWORKS | IDE PRODUCTIVITY TOOLS

Devexpress™

Building Apps with HTML5: What You Need to Know

Brandon Satrom

HTML5 is here, and the Web will never be the same.

You've no doubt heard that before, or something like it. I'd guess that when you did, you got excited, rolled your eyes, or mouthed the word "why?" and furrowed your brow a bit. Perhaps your reaction was a mix of all three.

I wouldn't blame you for any of these. HTML5 is exciting, and it does have the potential to change the Web as we know it, but it also gets blown out of proportion. What's more, its true meaning can be elusive. I've experienced each of those reactions myself while building applications with HTML5. It's a broad topic, so it's difficult to wrap your head around HTML5, much less know where to begin with this exciting new set of technologies.

This is the first article in a series for *MSDN Magazine*, and the goal is to give you a complete picture of why the first sentence in this article is true—and important. Over the next several months, I want to help you understand what HTML5 means to you—both as a Web developer and as a developer who uses Microsoft tools and technologies. I hope to simplify some of the complexity around HTML5 for you, and demystify much of the hype. I'll also introduce some HTML5 features that are available today, as well as some exciting technologies that, though a bit further out, are worth paying attention to. Finally, I'll leave you with some tips that will

This article discusses:

- What HTML5 is
- Microsoft's approach to HTML5
- Microsoft development tools for working with HTML5
- Evaluating HTML5 technologies for your applications

Technologies discussed:

Internet Explorer, WebMatrix, Visual Studio 2010,
Expression Web 4, Modernizr

help you adopt HTML5 technologies now, while you continue to provide great experiences to users with older browsers.

If you're excited about HTML5, I want to help you turn that excitement into ideas you can put into practice immediately. If you're skeptical, I want to help you understand just why HTML5 is important. And if you're just confused about what HTML5 even means, fear not: that's our first stop in this series.

What Is HTML5?

You might have discovered by now that HTML5 means different things to different people. To some, it just means new tags like `<header>` and `<footer>` and a handful of new attributes available in markup. To others, it means everything that's new and interesting on the Web, including technologies implemented in just a single browser or other specifications not officially part of HTML5. To be sure, understanding the real meaning of HTML5 is often the first roadblock many of us face.

And, honestly, there's some justification for the number of varying definitions. HTML5 is huge! Formally defined by an international standards body known as the World Wide Web Consortium (W3C), HTML5 consists of more than 100 specifications that relate to the next generation of Web technologies. By putting all 100-plus of these specifications under the moniker HTML5, you could argue that the W3C oversimplified things. And while it's hard to take something as broad as HTML5 and define it in an unambiguous way, I believe that the W3C was trying to address the scope of what's changing on the Web by introducing HTML5 as a unifying concept for that change.

In fact, HTML5 is an umbrella term describing a set of HTML, CSS and JavaScript specifications designed to enable developers to build the next generation of Web sites and applications. What's notable in that definition is its three parts: HTML, CSS and JavaScript. They define how developers use improved markup, richer style capabilities and new JavaScript APIs to make the most of new Web devel-

opment features. Simply put, HTML5 = HTML + CSS + JavaScript.

And that's it. HTML5 is about changes to HTML, CSS and JavaScript. Rather than worrying about all 100-plus specifications, those three terms describe the breadth and scope of HTML5. Still think that's a bit simplistic? It may be, but as you'll soon see, a comprehensive definition of HTML5 doesn't matter as much as the technologies you choose as worthy of your time and effort to adopt.

With a definition in hand, let's spend a few moments talking about where Microsoft fits into the HTML5 space.

HTML5 and Internet Explorer

As I mentioned, the set of specifications that make up HTML5 are stewarded by the W3C. The W3C consists of staff, organizations and individuals invested in helping to drive and define the future of the Web. The WC3 is a consensus-based organization, and typically operates by forming committees (called working groups) to divide up chunks of work on related specifications. Specifications can be proposed by any member, and all specifications owned by the W3C—more specifications than those that fall under the HTML5 umbrella—move through a five-stage process from first draft to official recommendation.

Microsoft is a member of the W3C and plays a very active role in the specification process for many HTML5 standards and working groups. Just like all of the major browser vendors, Microsoft is heavily invested in HTML5 and is working with the W3C and other vendors to ensure that developers can count on HTML5 technologies being reliably implemented in an interoperable way on all major browsers.

In the context of Microsoft the browser vendor, the approach is fourfold:

1. Deliver the best site-ready HTML5 today via Internet Explorer 9.
2. Expose upcoming features to developers via Internet Explorer Platform Previews.
3. Invest in interoperability through tests submitted to the W3C.
4. Prototype unstable standards via HTML5 labs.

“Site-Ready HTML5” is the term Microsoft uses to describe HTML5 technologies that you can use today because they have broad support across all major browsers. Technologies like the new HTML tags, Canvas, Scalable Vector Graphics, Audio and Video, Geolocation, Web Storage and many new CSS3 modules all fall into this space, and they're implemented in Internet Explorer 9, as well as the other mainstream browsers. We'll spend a fair amount of time in this series discussing these technologies, as well as how you can adopt them today.

Beyond what's available at present, Microsoft is using public Platform Previews to inform developers of what's coming in the next version of the browser, as well as to gather feedback. For Internet Explorer 9,

Microsoft released Platform Previews every six to eight weeks, each time announcing new HTML5 enhancements, features and performance improvements for developers to try out and evaluate. Internet Explorer 9 was released in March and as of early July, Microsoft has released two Platform Previews for Internet Explorer 10, signaling that Microsoft is continuing a regular release cadence for Internet Explorer previews. As a developer, you'll want to take advantage of the latest previews to learn, test and influence how the browsers evolve. You can download the latest Internet Explorer Platform Preview at IETestDrive.com.

To ensure that HTML5 works consistently across all browsers, Microsoft has invested heavily in interoperability, creating and submitting the single largest suite of test cases related to HTML5 to the W3C. For the first time, this suite of test cases will be used by the W3C as the authoritative source of HTML5 “readiness” in each browser. The end result for you and me as developers is that we can adopt and implement HTML5 technologies once, and trust that they'll work consistently across all browsers. For more information on Microsoft's work around interoperability, go to bit.ly/dxB12S.

While some HTML5 technologies already exist in Internet Explorer 9, and others are being announced for Internet Explorer 10 via Internet Explorer Platform Previews, some popular and newsworthy specifications need a bit more work by the W3C and the browser vendors before they'll be ready to implement in our applications. One such example is Web Sockets, an exciting specification that lets developers open bidirectional communication channels with back-end servers, thus enabling a level of “real-time” connectivity not previously available in Web applications. As a developer, you can no doubt imagine countless uses for Web Sockets in the applications you're building right now. But the Web Sockets specification is still changing at a rapid pace, with key aspects still in flux and being discussed within the W3C. Given that situation, it would be difficult to provide this feature consistently and reliably across all browsers today.

For unstable or evolving specifications like Web Sockets (which we'll cover in depth in a future article), Microsoft created HTML5

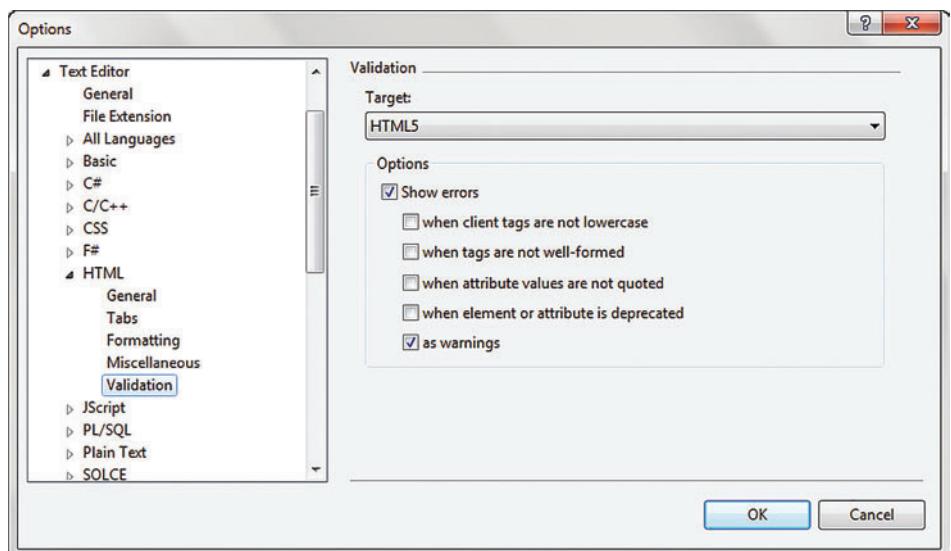


Figure 1 Enabling the HTML5 Schema via the Options Dialog

Labs, a site for developers to experiment with draft implementations of these technologies. The site provides prototypes you can download and try locally, as well as hosted demos for some specs. The goal is to give you a place to try these specs out for yourself, and for you to give both Microsoft and the W3C feedback on these specs as they stabilize and near implementation in browsers. For more information on HTML5 Labs, go to html5labs.com.

HTML5 and Microsoft Developer Tools

Beyond Microsoft's involvement with the W3C and the HTML5 technologies supported in the browser, there's another dimension to Microsoft's approach to HTML5 that's important for developers: its approach to HTML5 tooling.

In early 2011, Microsoft updated two of its development tools with service packs: Visual Studio 2010 and Expression Web 4. The service packs for both of these tools provided an HTML5 document type for validation, as well as IntelliSense for new HTML5 tags and attributes. If you're using Visual Studio 2010 SP1, you can enable the HTML5 Schema by clicking Tools | Options | Text Editor | HTML | Validation, and then selecting the HTML5 option in the Target drop-down list, as shown in **Figure 1**. You can also set HTML5 as the default schema from the HTML Source Editing Toolbar in any HTML file, as shown in **Figure 2**.

Once your default schema is set, you'll gain IntelliSense support in Visual Studio for the 28 new semantic tags in HTML, as well as new tag-specific and global attributes, as shown in **Figure 3**.

Microsoft further updated its HTML5 support with its release of the Web Standards Update for Microsoft Visual Studio 2010 SP1 in June 2011. This extension, which works with all editions of Visual Studio 2010, adds further HTML5 IntelliSense and validation to Visual Studio, includes JavaScript IntelliSense for new browser capabilities like Geolocation and DOM Storage, and provides comprehensive CSS3 IntelliSense and validation. You can download this extension, which will be regularly updated to provide enhanced tooling for HTML5 development, from bit.ly/m7OB13.

For Expression Web 4 SP1, setting the HTML5 schema under Tools | Page Options offers the same IntelliSense, and the tool also provides CSS3 IntelliSense for several draft CSS3 modules like border-radius, box-shadow, transform and the like.

If you're using WebMatrix (see web.ms/ WebMatrix), you may have noticed that all new .html, .cshtml or .vbhtml documents you create contain default markup similar to what's shown in **Figure 4**. As I'll discuss in the next article in this series, this is a basic, valid HTML5 document. Most notably, the doctype and meta charset tags have lost a lot of cruft. Using this simple doctype triggers HTML5 mode across all modern browsers, and WebMatrix makes it easier for you by providing an HTML5 document by default.

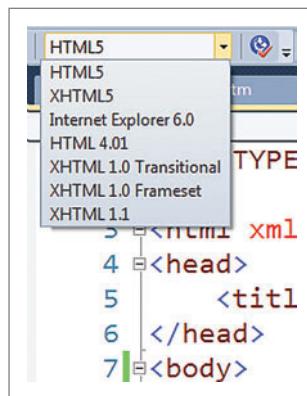


Figure 2 Setting the HTML5 Schema on the HTML Source Editing Toolbar

If that's not enough new HTML5 tooling for you—all since January 2011, by the way—ASP.NET MVC recently got in on the fun with the ASP.NET MVC 3 Tools Update announced at MIX11 in April. Along with a number of other great new tooling features, the ASP.NET MVC 3 Tools Update provides the option to use the HTML5 doctype for new projects—and ships Modernizr 1.7 in the Scripts folder of new applications. Modernizr is a JavaScript library that greatly eases HTML5 development; I'll discuss it in depth in a future article.

The takeaway here is that even though HTML5 is just emerging in our browsers, official tool support is quickly being added, and Microsoft is even adding support for libraries (like Modernizr) from the community. You can target HTML5 with some help from Microsoft tools today, and expect that that HTML5 support will continue to grow and improve over time.

'Adopting' HTML5 in Your Applications

By now, you should realize that HTML5 isn't a single entity that you can adopt or migrate to in one fell swoop. Adopting HTML5, rather than being a wholesale choice, is about making a technology-by-technology evaluation and determining which technologies are right for your application. For each HTML5 technology you evaluate, look at (at least) the following factors when deciding whether that technology is ready for you to adopt:

1. How widely implemented across all major browsers is the technology?
2. How would you adopt this technology and "polyfill" support for browsers that don't support a given feature?

The first factor is the most important, and when combined with an understanding of the browsers commonly used by visitors to your site, should give you a clear picture of which subset of the 100-plus specifications is worth evaluating further. That subset should consist of a set of stable specifications you can reliably adopt today for your users.

However, even with that stable set of HTML5 technologies, you shouldn't ignore your users who haven't moved to a newer browser. If you're heavily involved in the day-to-day development for your site, you no doubt have some rough idea of the percentages

of users visiting your site with a given browser. For most of us, it would be easy to look at the percentage of users visiting with an older browser and come to the conclusion that adopting any HTML5 technologies would negatively impact those users. Luckily there's "polyfilling" to save us from waiting until some foggy date in the future to adopt HTML5.

Paul Irish (a developer on the jQuery and Modernizr projects) defines a polyfill as "... a shim that mimics a future API, providing fallback functionality to older browsers." A polyfill is like spackle for

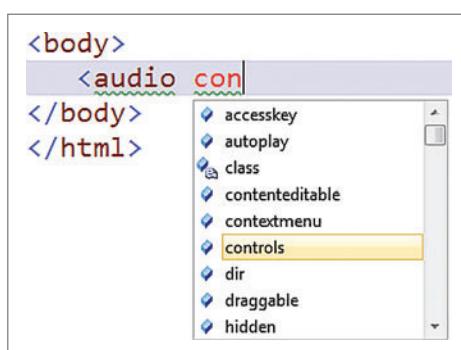


Figure 3 HTML5 IntelliSense in Visual Studio 2010 SP1

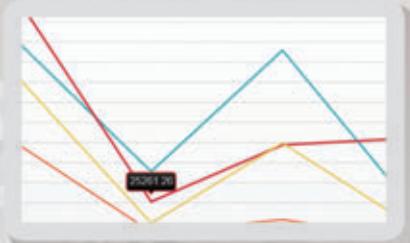
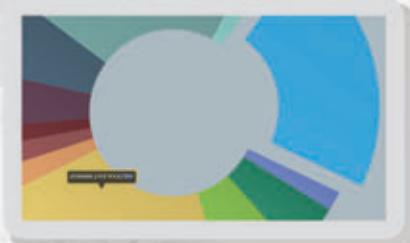
MEET THE NEW WEB STACK

From WebForms to MVC

Our new web stack is the ultimate kit for web development. We have tools that range from WebForms to MVC and from pure client-side to robust server-side development all powered by our core technology: Wijmo.



Number	Player	Age	Birthplace	Position
27	Canada Adams, Craig	32	Santa, Brazil	RW
45	Canada McLean, Michael	36	Saint-Augustin, Q	R
24	Canada Phillips, Phillip	36	Ottawa, Ontario	SW
21	Canada Cooke, Matt	36	Kitchener, Ontario	LW
20	Canada Cook, Sidney	24	Calgary, Alberta, Canada	CW
1	United States Cory, John	25	Shreveport, Louisiana	R
9	Canada Dudas, Pascal	30	Montreal, Quebec	W
7	United States Eaton, Mark	32	Wilmington, Delaware	D
26	Ukraine Fashchyns, Ruslan	36	New U.S.R.	LW
29	Canada Fleury, Max-Antoine	24	Sainte-Quitterie, Q	R
89	Portugal Pacheco, Henrique	24	Porto, Portugal	SW

ComponentOne®
Studio for MVC **wijmo**®



Wijmo Scaffolding in MVC
plus Client-side jQuery UI Widgets

ComponentOne®
Studio for ASP.NET **wijmo**®



Full-featured ASP.NET Server-side Controls
plus ASP.NET Ajax Extender Controls

Visual Studio® **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

DECEMBER 5-9
ORLANDO, FLORIDA



SAVE THE DATE! VISUAL STUDIO LIVE! COMES BACK TO ORLANDO

INTENSE TRAINING + AN AWESOME LOCATION:

that is the Visual Studio Live! experience. You'll get one-on-one access to industry pros over 5 days of real-world education on:

- ▶ **Visual Studio 2010 / .NET**
- ▶ **Data Management**
- ▶ **Silverlight / WPF**
- ▶ **Windows Communication Foundation**
- ▶ **Web / HTML5**
- ▶ **Cloud Computing**
- ▶ **Windows Phone 7**
- ▶ **Programming Practices**

**REGISTRATION
IS NOW OPEN.**

**REGISTER
TODAY TO
SAVE \$300!**

ALL-INCLUSIVE CONFERENCE + WORKSHOP + HOTEL PACKAGES
WITH ACCOMMODATIONS AT UNIVERSAL STUDIOS ROYAL PACIFIC RESORT ARE AVAILABLE!

VSLIVE.COM/ORLANDO

PLATINUM SPONSOR



SUPPORTED BY



PRODUCED BY

your Web sites; it's a way to determine if a given HTML5 feature is available to the user currently browsing your site, and to provide either a shim that "fills in" that support or a course of graceful degradation that enables your site to still function fully.

The most popular library associated with polyfilling is Modernizr, the JavaScript library I mentioned earlier. Modernizr provides some basic polyfills for semantic markup, feature detection for major HTML5 technologies and support for conditional CSS based on supported features. As noted, Modernizr will be the subject of an upcoming article; it will also feature prominently (along with many other polyfilling libraries) throughout this series. To learn more, download Modernizr at modernizr.com.

When it comes to choosing which technologies to adopt, your final list may be a combination of widely supported specifications and other specifications for which you'll have to polyfill support for certain browsers. Only you will know the exact makeup of that list based on your current needs and context.

In the coming months, I'll discuss several notable specifications, from Geolocation and Forms and Canvas, to Web Workers, Web Sockets and IndexedDB. Some of these are widely supported and "site-ready," and some, like Web Sockets, are too groundbreaking to ignore, regardless of where they stand today. With each specification, I'll discuss current and known future support, some basics about how you can implement the specification's features on your sites, and how to polyfill support for browsers that don't support a given feature.

If you want to dig more into HTML5 today, I suggest you pick up a couple of books on the subject. In particular, I recommend "Introducing HTML5" (New Riders, 2010) by Bruce Lawson and Remy Sharp and "HTML5 Up and Running" (O'Reilly Media, 2010) by Mark Pilgrim. Also, be sure to visit W3C.org for up-to-date information on all specifications, as well as BeautyoftheWeb.com and IETestDrive.com to download Internet Explorer 9 and the Internet Explorer 10 Platform Preview, respectively, and learn more about the great HTML5 experiences Microsoft is delivering through the browser.

Figure 4 A Default HTML Document in WebMatrix

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title></title>
  </head>
  <body></body>
</html>
```

Above all else, start adopting HTML5 today. The Web won't ever be the same, really, and you can be part of the catalyst by building the next great Web applications using HTML5. ■

BRANDON SATROM works as a developer evangelist for Microsoft outside of Austin. He blogs at UserInexperience.com and can be followed on Twitter at Twitter.com/BrandonSatrom.

THANKS to the following technical experts for reviewing this article:
Jon Box, Damian Edwards and Clark Sell

**we are Countersoft
you are Control**

I am agile
I am testing
you are bug tracking
I am issue tracking
I am job management
I am answers
I am product support
you are software development
I am project management
I am creating
I am help desk

GEMINI
Project Platform

**BOOK A DEMO
FREE TRIAL**
www.geminiplatform.com

The most versatile project management platform for software development and testing teams around the world.

Allows teams to work the way they want to work with more functionality and easy customization for optimum team performance.

ATLAS
Product Support Optimized

DOWNLOAD NOW
and lead from the front
www.atlasanswer.com

Because in product support you should only answer any question once. Knowledge to the people!

A new generation of community support and self-help software. Integrates Q&A, Knowledge Base, FAQs, Documents, Videos and Podcasts and a simple, feature-rich User Interface.

COUNTERSOFT
ENABLING
COLLECTIVE
CAPABILITY

Europe/Asia: +44 (0)1753 824000 // US/Canada: 800.927.5568
sales@countersoft.com www.countersoft.com

Build a Ticketing System Using Exchange and Team Foundation Server

Mohammad Jalloul

Team Foundation Server(TFS) provides a great mechanism for creating and managing tickets via its powerful work item tracking functionality. Such tracking is typically accomplished in one of two ways. In the first, a support representative with access to TFS via its Web interface (known as Team System Web Access, or TSWA) creates a custom work item type for a support ticket, entering the necessary details. In the second method, an interface is created to wrap around the work item functionality and simplify access to it for external users who don't need to know about TFS. The work item is then typically assigned to a software engineer who tracks and manages the work item from within Visual Studio. The goal of both methods is to move ticket creation from e-mail or proprietary database systems to TFS due to its many advantages—in particular

that TFS work items have a workflow that defines the lifecycle of a ticket. Anyone can keep track of a ticket by monitoring changes to the work item and viewing the work item history. Integration with the TFS data warehouse provides an added benefit, as you can leverage the powerful reporting capabilities of SQL Server Reporting Services (SSRS) or Excel Services.

Despite such methods, e-mails often get sent around anyway, to further explain the issue or add more detail. This additional e-mail-thread information is sometimes lost, and other times manually copied into the work item history to keep the context of the work item as complete as possible. This manual intervention defeats the purpose of work items and can kill the productivity that work items were designed to provide.

I'll present a third method in this article that takes the best of both worlds: e-mail and work items. This approach starts with e-mail—just as support reps have been doing for a long time.

This article discusses:

- Creating a Web service to listen to notifications from Exchange
- Establishing a subscription in Exchange for the Web service
- Deploying the Web service
- Configuring a work item using TFS Power Tools
- Customizing the PushNotification SDK sample

Technologies discussed:

Exchange Web Services 1.1, Team Foundation Server 2010, Team Foundation Server Power Tools 2010, Exchange Server 2010, Visual Studio 2010

Code download available at:

code.msdn.microsoft.com/mag201108TFS

Exchange Web Services

The process I'll present uses the power of Microsoft Exchange Web Services (EWS) to subscribe to notifications for e-mails sent to an Exchange 2007/2010 distribution list. The listening Web service creates a work item based on any new e-mail received. Any subsequent e-mails related to the same thread are appended to the work item history, using the conversation-tracking capabilities of Microsoft Exchange Server, thus mirroring the e-mail thread in the work item. The work item also maintains a workflow. This way the users—the support representatives—are not changing their habits to adapt to a new technology; instead, the technology is taking their process and automating it.

Figure 1 shows how the process will typically flow.

The following explains the flow in more detail:

1. The flow starts when a support representative is notified of an issue with a product.
2. The support rep sends an e-mail describing the issue to a distribution list (DL) in Exchange. The DL includes anyone who needs to know about the issue and can assign it to the appropriate developer to resolve it. The DL also contains a “watcher” e-mail account that exists only for notification purposes. The watcher account is registered with Exchange so that it will receive notifications at a specific Web service address.
3. Exchange processes the e-mail, routing it to all the members of the DL and sending notifications to any Web services that are registered to receive them for a specified e-mail account.
4. The support ticket Web service receives a notification from Exchange via its Web service interface.
5. The Web service processes the notification and requests additional information related to the e-mail address. It then creates a Support Request work item in TFS, which is populated with information extracted from the e-mail message.
6. The DL members also receive the e-mail and they may respond by asking for more information, which the support rep will send. Because the DL is included in the To or CC fields of the e-mail message, the watcher e-mail account is aware of this and the notifications Web service receives notifications of all the e-mails happening in the thread. TFS updates the Support Request work item with the additional information.
7. The ticket is assigned to a developer using the Assigned To field of the Support Request work item just created.
8. The developer starts working on the issue, changing the state of the work item to In Progress. When the code containing the fix is reviewed, tested and completed, it will be checked in. The resulting changeset is associated with the Support Request work item. This work item now tracks both the e-mail conversation and the changeset related to the fix.
9. If the support rep has access to TFS, the Support Request can be assigned to him for closing. Otherwise, the DL can be notified via e-mail. Typically, project managers and leads will set up TFS Alerts so they’re notified when a Support Request work item has been resolved. This takes place when the work item is changed from In Progress to Done.

The process is actually similar to what usually takes place in real-life scenarios involving support issues—except, in real life, the e-mail thread and ticketing system are typically two completely different and separate systems.

The Support Ticket Notifications Web Service

The Support Ticket Notifications Web service uses the EWS Managed API, which lets managed .NET applications communicate with EWS using documented interfaces via familiar SOAP-based messaging. You’ll find a detailed description of the EWS Managed API at bit.ly/jGKRgG.

EWS is not a typical part of a Microsoft Exchange installation, and in many IT setups, it’s installed on a separate Exchange Client

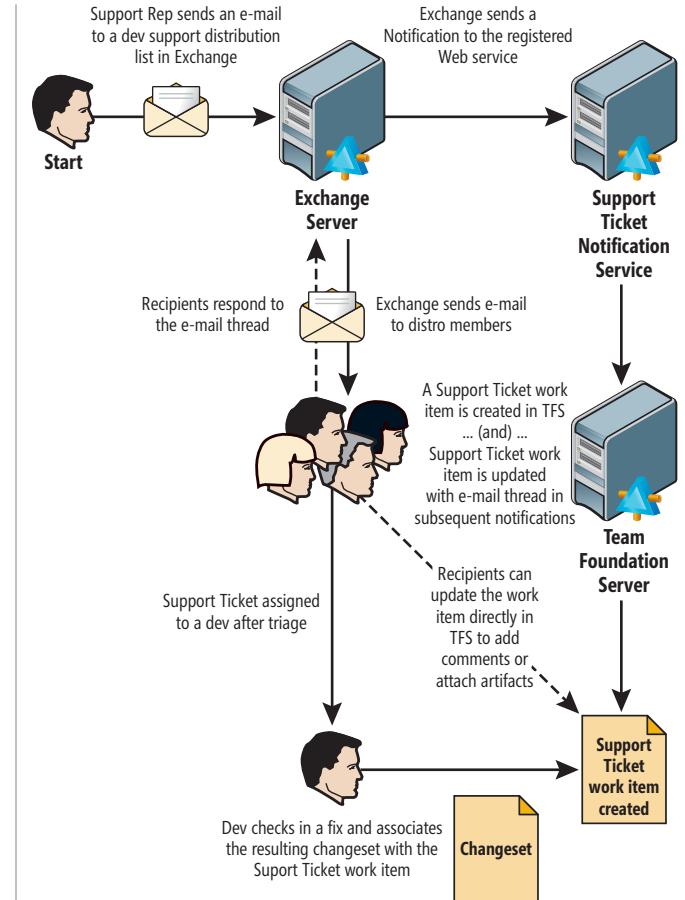


Figure 1 Process Flow for the Support Ticket Work Item

Access server. To be usable, the services have to be installed and configured with Microsoft Exchange. You’ll need to obtain the EWS Web service URL from your Exchange administrators, which will look something like <https://hostname/EWS/Exchange.asmx>. Note that these Web services are typically set up on an SSL endpoint.

The latest version of the EWS Managed API at the time of this writing is 1.1. This version supports Microsoft Exchange Server 2007 SP1 through Exchange Server 2010 SP1. You can download it from bit.ly/mkb7ls.

EWS is not a typical part of a Microsoft Exchange installation.

By default, the installer copies the files to [Program Files]\Microsoft\Exchange\Web Services\1.1. The API is packaged into a single assembly named Microsoft.Exchange.WebServices.dll. The installation folder includes two important files you should review before starting to work with the API: Readme.htm file and GettingStarted.doc.

The first step in interacting with EWS is to create a Web service to listen to notifications coming from Exchange. Once this Web service is created, it must be registered with Exchange in order to start receiving notifications.

Figure 2 Subscription Code from the PushNotification Sample

```

public static void SubscribeForPushNotifications()
{
    System.Net.ServicePointManager.ServerCertificateValidationCallback =
        delegate(Object obj, X509Certificate certificate, X509Chain chain,
        SslPolicyErrors errors)
    {
        // Replace this line with code to validate server certificate.
        return true;
    };

    // Create the bindings and set the credentials.
    ExchangeServiceBinding esb = new ExchangeServiceBinding();
    esb.Url = "https://CAS01.contoso.com/EWS/exchange.asmx";
    esb.Credentials = new NetworkCredential("username", "password", "domain");

    // Create a new subscription.
    SubscribeType subscribeRequest = new SubscribeType();
    PushSubscriptionRequestType pushSubscription =
        new PushSubscriptionRequestType();

    // Subscribe to events in the inbox folder.
    BaseFolderIdType[] folders = new BaseFolderIdType[1];
    DistinguishedFolderIdType folderId = new DistinguishedFolderIdType();
    folderId.Id = DistinguishedFolderIdNameType.inbox;
    folders[0] = folderId;
    pushSubscription.FolderIds = folders;

    // Subscribe to all events.
    NotificationEventType[] eventTypes =
        new NotificationEventType[6];
    eventTypes[0] = NotificationEventType.NewMailEvent;
    eventTypes[1] = NotificationEventType.CopiedEvent;
    eventTypes[2] = NotificationEventType.CreatedEvent;
    eventTypes[3] = NotificationEventType.DeletedEvent;
    eventTypes[4] = NotificationEventType.ModifiedEvent;
    eventTypes[5] = NotificationEventType.MovedEvent;
    pushSubscription.EventTypes = eventTypes;

    // Receive push notifications every 1 minutes.
    pushSubscription.StatusFrequency = 1;

    // Identify the location of the client Web service.
    pushSubscription.URL = "http://clientWebService/Service.asmx";

    // Form the subscribe request.
    subscribeRequest.Item = pushSubscription;

    // Send the subscribe request and get the response.
    SubscribeResponseType subscribeResponse =
        esb.Subscribe(subscribeRequest);

    // Check the result.
    if (subscribeResponse.ResponseMessages.Items.Length > 0 &&
        subscribeResponse.ResponseMessages.Items[0].ResponseClass ==
        ResponseClassType.Success)
    {
        SubscribeResponseMessageType subscribeResponseMessage =
            subscribeResponse.ResponseMessages.Items[0] as
            SubscribeResponseMessageType;

        using (StreamWriter sw = new StreamWriter("MailboxEventLog.txt"))
        {
            sw.WriteLine("Subscribed for Push notifications: {0}",
            subscribeResponseMessage.SubscriptionId);
        }
    }

    CreateXmlMessageTextFile(subscribeRequest, subscribeResponse);
}

```

The Microsoft Exchange Server 2010 SP1 Web Services SDK simplifies the creation of such Web services. You can download the October 2010 release (the latest version at the time of this writing) at bit.ly/kZtjLq.

You'll need an Exchange account to create a notifications subscription with Exchange.

The SDK makes it much easier to build the first Web service. It contains four samples that explore different areas targeted by the API. You can access the samples after installing the SDK by navigating to the Start menu | All Programs | Exchange Server 2010 SP1 Web Services SDK | October 2010 | Exchange Server 2010 SP1 Web Services SDK Sample Directory. The sample that's of particular interest is PushNotification, which demonstrates a push notifications Web service that listens to and processes incoming notifications from Exchange. This sample also contains a client application you can use to create a subscription in Exchange for the Web service. Using this sample as a base implementation makes it easy to understand the way EWS works and to verify that all the required pieces are functional.

Subscribing with Exchange

You'll need an Exchange account to create a notifications subscription with Exchange. The subscription will set up the routing such that any e-mails sent to that account's mailbox will result in Exchange

making a call to the notifications service registered for that account. Note that it's possible to register more than one notification for any given account. In fact, it's even possible to create more than one subscription for the same Web service endpoint. From an Exchange and EWS perspective, this is perfectly legitimate.

The only requirement for Web services registered with EWS is that they implement a certain interface, `INotificationServiceBinding`, which has the following signature:

```

public interface INotificationServiceBinding {
    SendNotificationResultType SendNotification(
        SendNotificationResponseType sendNotification);
}

```

The only requirement is that the Web service contains an implementation of the `SendNotification` method. This is the Web method

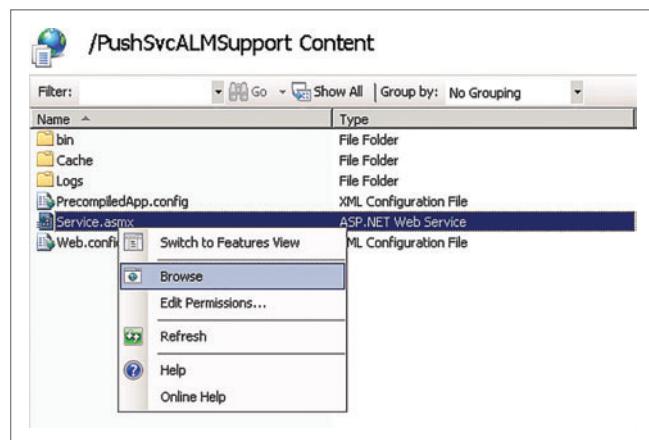


Figure 3 Browsing to the ASMX Page

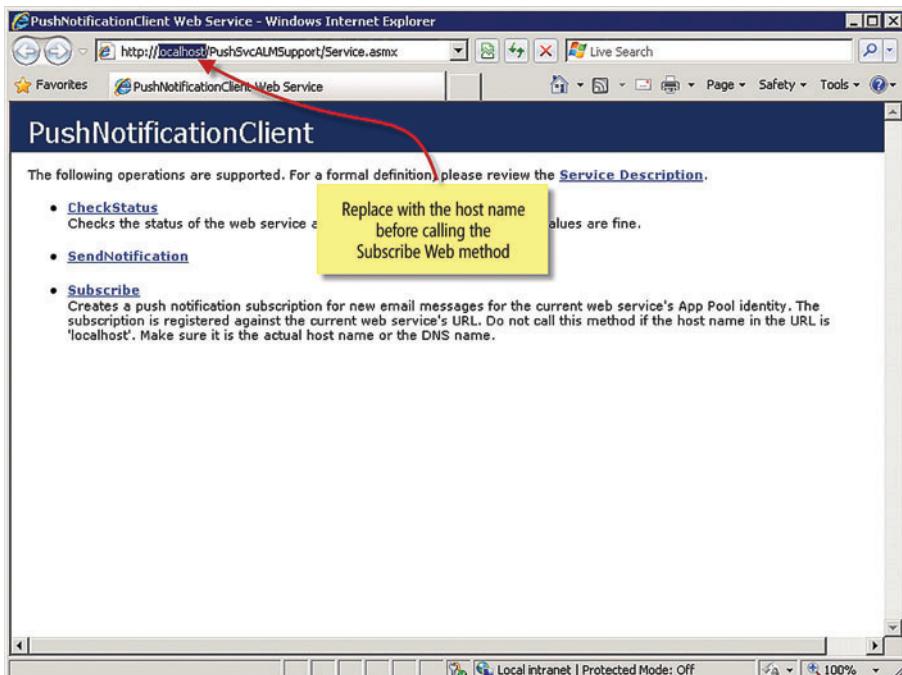


Figure 4 Default Web Service URL with localhost as the Host

that will be called by Exchange whenever an e-mail matching the subscription attributes is received.

Going back to the PushNotification sample and, in particular, the PushNotificationSubscriber console application, it becomes clear what a subscription entails. **Figure 2** shows the `SubscribeForPushNotification` method taken from the `PushNotificationSubscriber.cs` file.

I've included the code in **Figure 2** here because it illustrates the key features available when registering a subscription. In particular, notice how the sample assigns the first piece of information, the Client Access server URL:

```
esb.Url = "https://CAS01.contoso.com/EWS/exchange.asmx";
```

It then uses the credentials of the e-mail account to authenticate with EWS:

```
esb.Credentials = new NetworkCredential("username", "password", "domain");
```

Notice also that once you've created a subscription object, you can configure it so that it only subscribes to certain events (new mail, copying, moving, deleting, modifying and so on) or to specific mail folders. This provides great flexibility, as you can delegate the filtering to Exchange rather than subscribing to all events and filtering in the application later.

Finally, note the `StatusFrequency` property. This is a numeric value that defines, in minutes, how often Exchange sends heartbeat calls to the Web service to make sure that the registered endpoint is alive. This helps Exchange cancel subscriptions to endpoints that are no longer valid, such as for a Web service that's moved to a new host and hence has a new URL. If you forget to unsubscribe before moving the Web service, the subscription can last

indefinitely, and Exchange would keep sending notifications unnecessarily to a nonexistent endpoint.

Now let's look at the customized implementation to support the creation of Support Request TFS work items. The best way to present the functionality is to start with a demo of the Web service and then dive into the implementation.

Deploying the Web Service

The code project that accompanies this article contains only a Web service and several other supporting artifacts. There's no separate console application to take care of the registration because it's built into the Web service.

Once built, the Web service can be deployed either from Visual Studio 2010 or by copying the resulting precompiled Web site to a host. You'll need to manually create the Web application and a corresponding application pool. Set the identity

of the application pool to the watcher user account. This is important because the Web service can impersonate the identity of the app pool and use that to perform the subscription. This has two benefits:

1. The credentials won't have to be re-entered every time the Web service needs to be reregistered.
2. The credentials won't need to be stored where they might be accessible in order to use them to resubscribe.

Once you've created a subscription object, you can configure it so it only subscribes to certain events.

The Web service needs to have write permissions to two folders: the folder to which it writes logs and the folder it uses as the TFS cache. In addition, you'll need to configure the following settings in `web.config`, as they depend on the particular environment where the Web service is used:

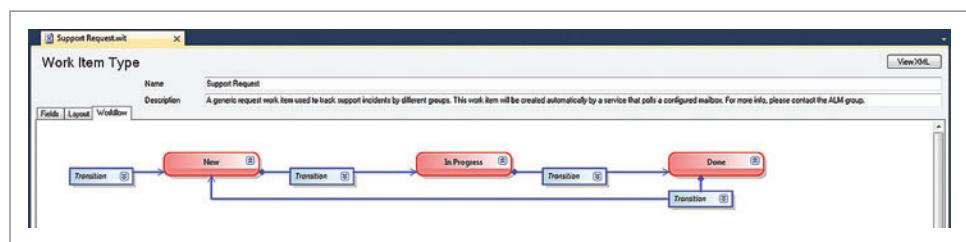


Figure 5 Workflow for the Support Ticket Work Item

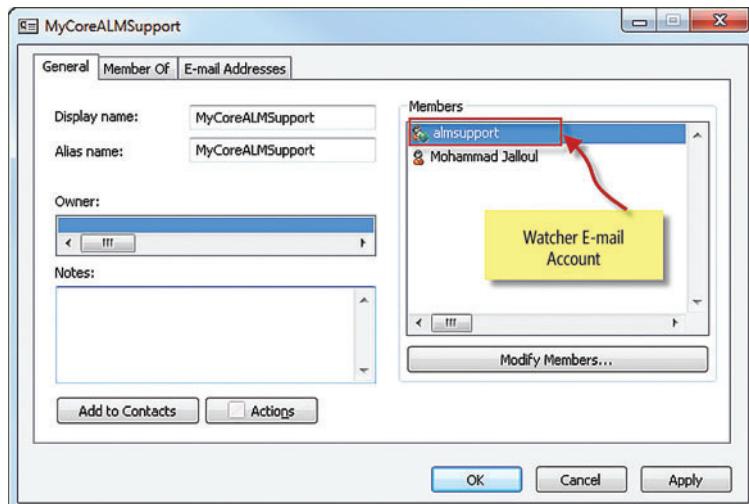


Figure 6 Adding the Watcher E-mail Account to a Distribution List

- **EWSServiceUrl:** The Exchange Service URL; get this URL from the Exchange administrators.
- **TeamFoundationServerUrl:** The URL of the TFS project collection where the work items are deployed (for example: <http://<servername>:8080/tfs/>).
- **WorkItemTeamProject:** The name of the TFS Team Project that contains the support ticket work item type (for example: SupportWorkFlow).
- **WorkItemType:** The actual name of the support ticket work item type (for example: Support Request). If the work item is renamed or if the Web service is used to support some other work item type (as I'll show later in this article), you can use this to set the name of the new work item.
- **RequestSupportTeamName:** The work item contains a "Request Support Team" field for directing different teams to different support tasks. This is driven by a global list called Teams in the work item definition.

To use this feature, keep the field definition and create the global list (instructions are included in the next section of this article, "Configuring the Work Item") and assign one of the values of the global list to the RequestSupportTeamName configuration key (for example: Customer Service). If not, remove the global list for the field and the "Required" constraint from the work item and leave the value of the RequestSupportTeamName key blank.

At this point the Web service is ready to be used. You can verify it's set up correctly using IIS Manager on the Web server to navigate to the Web application folder (remember

to click on the Content View tab at the bottom if using IIS 7 or later). Then right-click on the Service.asmx file and click on Browse, as shown in **Figure 3**. You should see a Web page identical to the one shown in **Figure 4**.

Before you can start using the Web service, you'll need to import the work item into TFS. This article assumes you're using TFS 2010, but the same work item and Web service can be used with TFS 2008.

Configuring the Work Item

The sample code for this article (which you can download at code.msdn.microsoft.com/mag201108TFS) contains a sample support ticket work item type called Support Request. In order to view the work item type definition, workflow and layout prior to importing into TFS, you need to install TFS Power Tools, available at bit.ly/hyUNqo. When you install TFS Power Tools, a new menu called Process Editor is added to the Visual Studio IDE under the Tools menu.

Double-click on Support Request.xml (located under the Work Item folder in the sample solution). The Work Item Editor window will be loaded with three tabs: Fields, Layout and Workflow, as shown in **Figure 5**.

The work item Workflow shows the different states that the work item supports: New, when the work item is first created; In Progress, when the issue documented in the work item is actually being worked on; and Done, when work on the issue concludes.

If you decide to configure the Request Support Team field, as discussed earlier, locate the Request Support Team field under the Fields tab, double-click it to open the Field Definition dialog, and then click on the Rules tab. If you don't plan on using this field, just delete the two rules listed (ALLOWEDVALUES and REQUIRED). If you do want to use the field but would like to configure it to use a different global list, double-click on the ALLOWEDVALUES rule and then on the only entry present: GLOBALLIST: Teams. In the List Item Edit dialog, pick a global list and click OK.

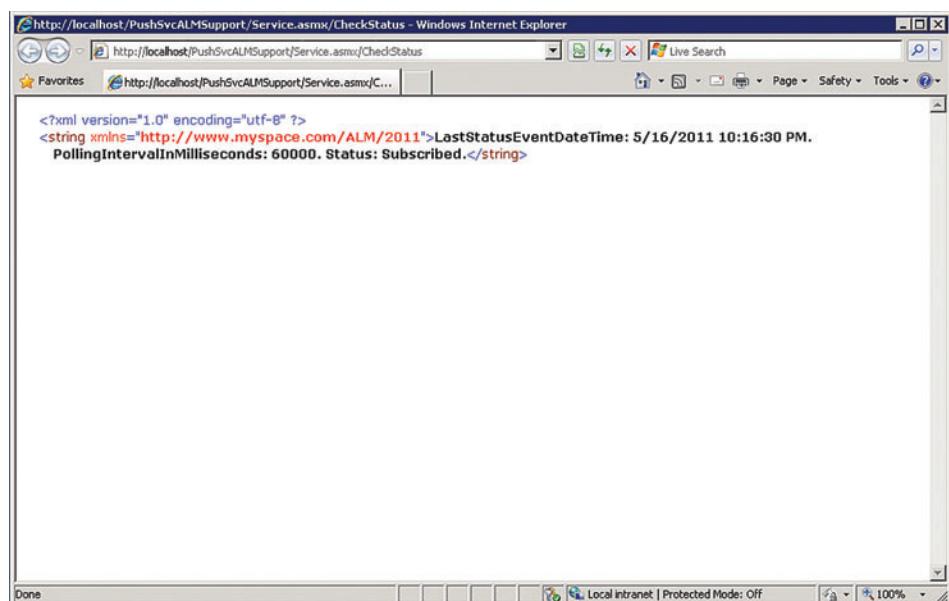
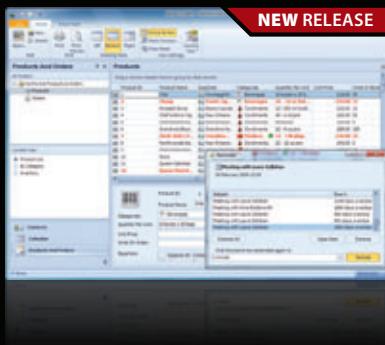


Figure 7 Sample Output from the CheckStatus Web Method



Janus WinForms Controls Suite V4.0 | from \$853.44



Add powerful Outlook style interfaces to your .NET applications.

- Includes Ribbon, Grid, Calendar view, Timeline and Shortcut bar
- Now features Office 2010 visual style for all controls
- Visual Studio 2010 and .NET Framework Client Profiles support
- Janus Ribbon adds Backstage Menus and Tab functionality as in Office 2010 applications
- Now features support for multiple cell selection



TX Text Control .NET for Windows Forms/WPF | from \$499.59



Word processing components for Visual Studio .NET.

- Add professional word processing to your applications
- Royalty-free Windows Forms and WPF rich 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



FusionCharts | from \$195.02



Interactive Flash & JavaScript (HTML5) charts for web apps.

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 18,000 customers and 375,000 users in 110 countries



ActiveAnalysis | from \$979.02



Rapidly embed out-of-the box OLAP, data visualization and BI features to your applications.

- Charts, pivot tables and data visualization in one programmable control
- Support for Silverlight, Windows Forms and ASP.NET development in one component
- Rich drag-and-drop user experience encourages self-discovery of data
- Excel exports allow end users to share analysis results offline
- Flexible data binding allows you to treat relational data as multi-dimensional

If you don't have a global list set up yet, you can create one (you'll need to have TFS Administrator permissions):

1. In the Visual Studio IDE, go to Tools | Process Editor | Global List and then click on Open Global List from Server.
- 2 You'll be presented with a dialog. Select the Team Project Collection and click on Connect.
3. In the Global List dialog, right-click on any item and then click on New Global List.

To use the notifications effectively, the watcher account should be part of a DL that is to be monitored.

4. Enter a name for the global list. You can keep the same name as in the work item: Teams.
5. Right-click the new global list and click on New Item. Repeat this step for as many times as needed to enter the team names.
6. Click OK to save the new global list.

Note that if you've given the new global list a different name, you'll need to edit the work item and update its definition with the global list name.

At this point, all that's left is to import the work item into TFS, like so:

1. In the Visual Studio IDE, go to Tools | Process Editor | Work Item Types and then click on Import WIT (WIT stands for Work Item Type).
2. Click the Browse button and navigate to the location of the Support Request.xml file.
3. In the Project to Import to list, click on the name of the TFS Team Project into which you want to import the work item and then click OK.

You can verify that the work item was successfully imported into TFS by right-clicking on the Work Items node for that Team Project from within Team Explorer and then expanding the New Work Item window. You should see Support Request as one of the available options. If you don't see it there, right-click the Work Items node and then click on Refresh.

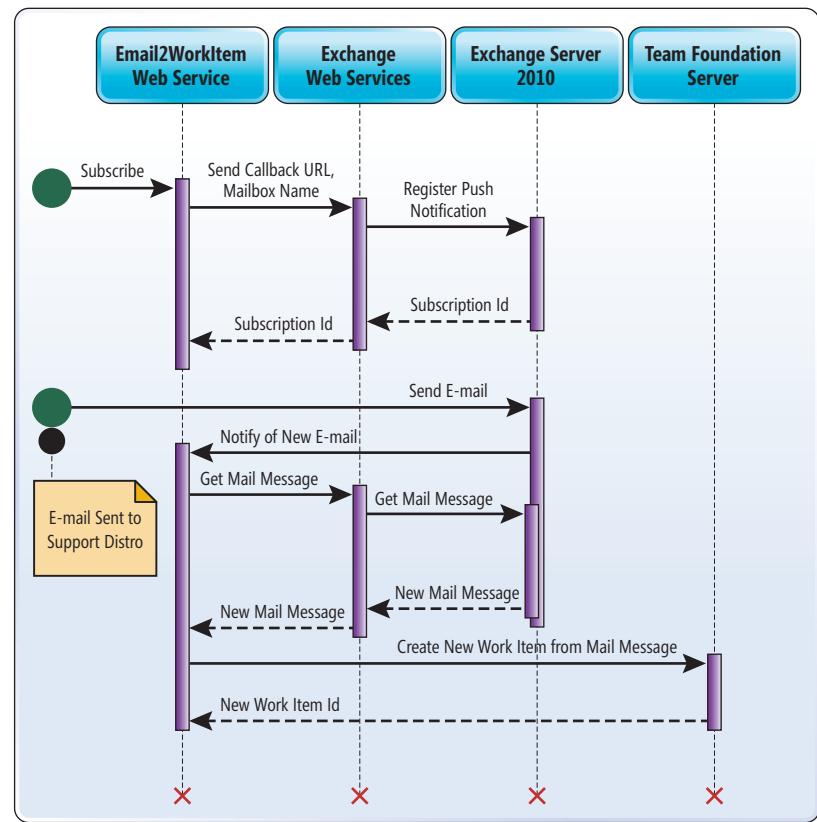


Figure 8 The Subscription and Notification Process

Creating a Subscription

As shown in [Figure 4](#), the Web service has three methods: Check Status, SendNotification and Subscribe. The Subscribe Web method is used to create the notification subscriptions.

The subscription process will create a subscription for the user account that's set as the identity of the Web service App Pool. To use the notifications effectively, that account—the watcher account—should be part of a DL that is to be monitored, as shown in [Figure 6](#).

To create a subscription, click on the Subscribe link on the Web page shown in [Figure 4](#). Then click on the Invoke button on the

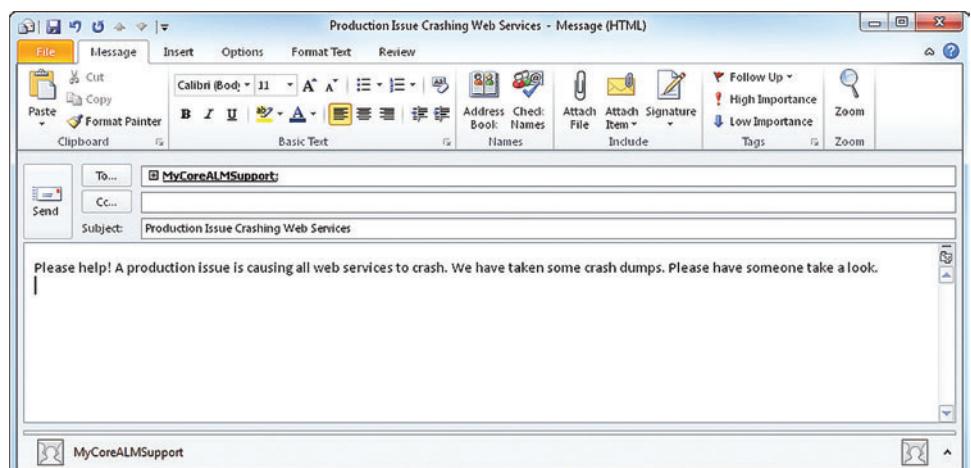


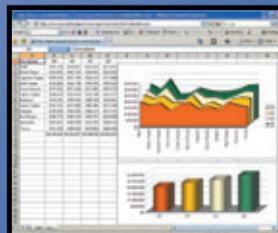
Figure 9 The First E-mail in a Mail Thread

In a League by Itself

"SpreadsheetGear 2010 is Fantastic! These new capabilities just propelled this control way-way past any competition, of which you have none IMHO. SpreadsheetGear is in a league all by itself."

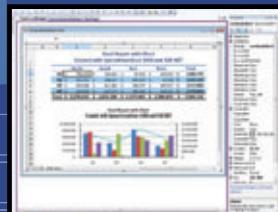
Greg Newman, Senior Software Engineer, WSFS Cash Connect

SpreadsheetGear 2010



ASP.NET / Excel / Reporting

Easily create richly formatted Excel reports without Excel using the new generation of spreadsheet technology built from the ground up for scalability and reliability.



Excel Compatible Windows Forms Control

Add powerful Excel compatible viewing, editing, formatting, calculating, charting and printing to your Windows Forms applications with the easy to use WorkbookView control.



Create Dashboards from Excel Charts and Ranges

You and your users can design dashboards, reports, charts, and models in Excel rather than hard to learn developer tools and you can easily deploy them with one line of code.

Download the FREE fully functional 30-Day evaluation of SpreadsheetGear 2010 today at

www.SpreadsheetGear.com.

 **SpreadsheetGear**

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

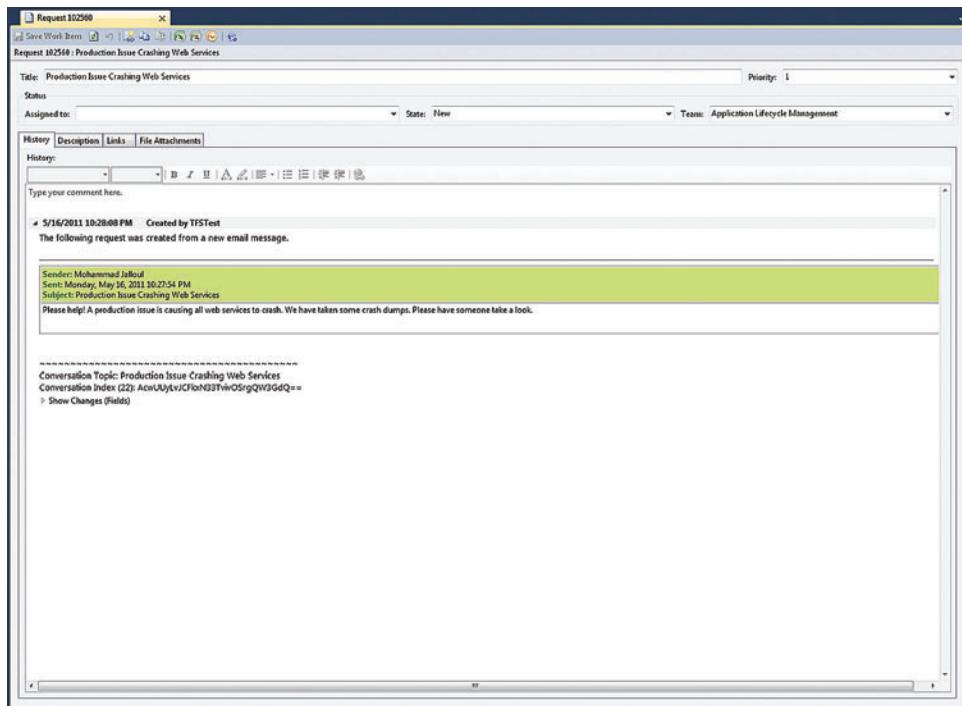


Figure 10 The Work Item Generated as a Result of the First E-mail

resulting Web page. Upon success, you'll see a subscription ID. This is also logged into the logs folder defined in the web.config file.

The details of the subscription are logged into two files. The first logs the request sent to EWS to perform the subscription and has the form `SubscribeType-<timestamp>.xml`. The second file logs the response received from EWS to confirm the subscription, and has the form `SubscribeResponseType-<timestamp>.xml`. A third

to the DL that contains the watcher account. **Figure 9** shows a sample e-mail to the DL shown in **Figure 6**, and **Figure 10** shows the Support Request work item that's generated as a result of the notification sent from Exchange to the notifications Web service.

Figure 11 shows the result of sending other e-mails as part of the same e-mail thread and how that's translated into the work item history. The first thread gets a distinct color for its header. The rest of the threads get another distinct color (yellow in this case, but the colors can be configured in web.config).

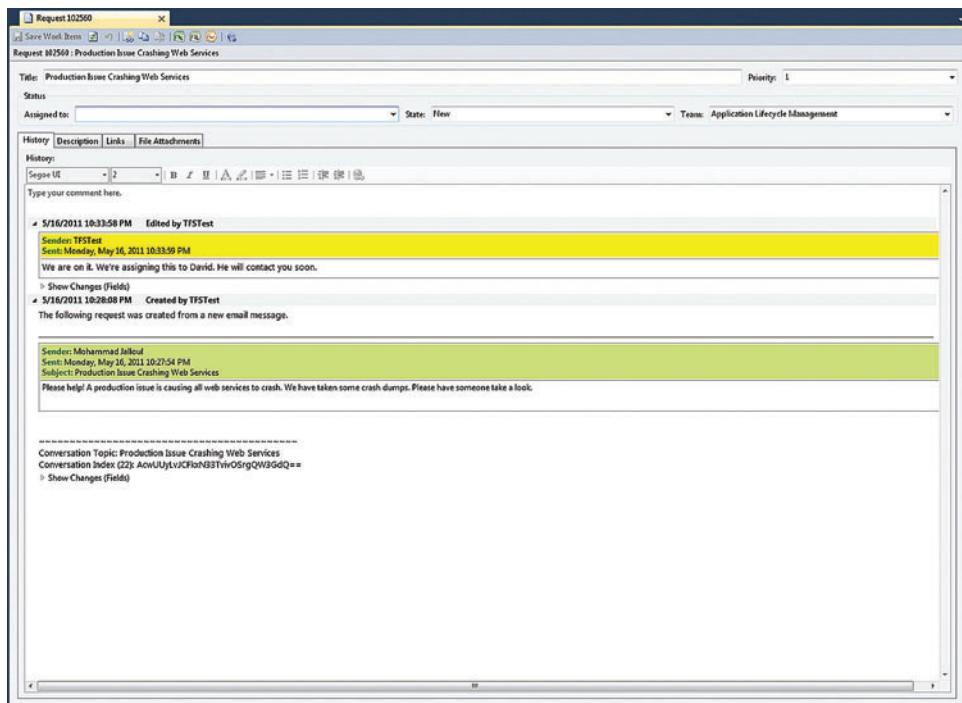


Figure 11 The Work Item After a Second E-mail

file named `SubscriberEventLog.txt` logs all the events received by the Web service from Exchange, including a heartbeat status event every minute and New Mail events whenever they're triggered.

The status of the subscription can be verified at any point by calling the `CheckStatus` Web method, which produces a result similar to that shown in **Figure 7**, which shows the time the last heartbeat event was received and the status of the Web service subscription. If there's an active subscription, you'll see "Status: Subscribed," as shown in the figure.

The overall process is depicted in the sequence diagram in **Figure 8**.

Using the Ticketing System

The easiest way to start using the ticketing system is to mimic a real scenario and send an e-mail

Customizing the Code

Now let's take a look at the different pieces of code that make the system operate. The sample code for this article contains a Visual Studio 2010 solution called `Email2Workitem.sln`. When you load this solution in Visual Studio, you'll see that the structure is divided into four solution folders: Diagrams, Subscription Library, Web Service and Work Item.

For the most part, the Web service is based on the PushNotification SDK sample. However, I made some changes in order to keep the solution generic and easy to use. I'll go over the changes and customizations I made to turn the sample into a ticketing solution.

We didn't invent the Internet...

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



applications

powered by 

connectivity

powered by 

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

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

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

Figure 12 The InitializeTimer Method

```
private void InitializeTimer()
{
    int autoSubscribeTimeoutThresholdInMinutes = Convert.ToInt32(
        ConfigurationManager.AppSettings["AutoSubscribeTimeoutThresholdInMinutes"]);

    lock (_pollingTimerSyncLock)
    {
        if (autoSubscribeTimeoutThresholdInMinutes > 0)
        {
            // Seed the timer with the polling interval specified in config.
            if (_pollingTimer == null)
            {
                TimerCallback timerCallback = OnTimedEvent;
                _pollingTimer = new Threading.Timer(timerCallback, null, 0,
                    _pollingIntervalInMilliseconds); // Enable the timer.
            }
        }
        else
        {
            // Dispose of the timer.
            if (_pollingTimer != null)
            {
                _pollingTimer.Dispose();
                _pollingTimer = null;
            }
        }
    }
}
```

The first major change is the functionality that lets the subscription process be integrated into the Web service. This is crucial because it allows the Web service to subscribe without having to manage the account credentials. Sometimes the subscription needs to be renewed, such as when the server hosting the Web service undergoes some maintenance activities, for example. When that happens, the App Pools are typically stopped and Exchange cancels the subscriptions because the heartbeat event gets no response. The auto-renewal of the subscription is achieved by calling the InitializeTimer method in the Web service constructor. InitializeTimer, shown in **Figure 12**, creates a timer with an interval specified in web.config. This timer is responsible for renewing a subscription after service interruptions, for example.

The second important change has to do with inspecting the incoming message to determine whether it's the first e-mail in a thread. **Figure 13** shows this functionality.

Notice how the code leverages the fact that the Web service App Pool identity is that of the mail account, which means that Web calls to retrieve the e-mail message details can be done without having to enter any credentials.

The ConvertEmailToRequestItem method does the bulk of the work. To determine whether an e-mail is the first e-mail or part of a thread, it makes use of the ConversationIndex property of the EmailMessage instance. The length of the ConversationIndex byte array is 22 when it's the first e-mail. Subsequent e-mails in a thread get further bytes added to the ConversationIndex. ConvertEmailToRequestItem uses this to determine whether to treat the e-mail as a new message and create a new work item for it, or as part of an existing thread, in which case it looks up the existing work item and appends the thread-specific part of the mail message to the work item revision history. Exchange exposes the part of an e-mail that's specific to the current e-mail in the thread via the EmailMessage UniqueBody property. To get the complete

thread, the Body property can be used. This is quite powerful, as it allows building the e-mail thread in the work item revision history, as shown in **Figure 11**.

One special case that's worth mentioning is when an e-mail arrives that's part of a thread but with no corresponding work item available to update. This typically happens if the thread was started before the ticketing system was put in place. In that case, a new work item is created.

WIQL gives you an easy yet powerful way to query a work item without having to load objects and enumerate through collections.

In order for the Web service to determine whether an e-mail already exists as a work item, it relies on a piece of information it stores in the work item when the work item is first created: ConversationIndex. The Web service queries work items based on that value to find out whether the work item exists. The querying is performed using the TFS work item query language (WIQL), as in the following code:

```
private const string QUERY_BY_CONVERSATION_INDEX_WIQL =
    "SELECT [System.Id] FROM WorkItems WHERE [System.TeamProject] = '{0}' AND
    [System.WorkItemType] = 'Support Request' AND
    [MsdnMag.SupportTicket.ConversationIndex] = '{1}' ORDER BY [System.Id] desc";
```

WIQL gives you an easy yet powerful way to query a work item without having to load objects and enumerate through collections.

The Web service also provides a way to easily track it in the TFS activity logs. The TFS activity logs are stored in two SQL Server tables (tbl_Command and tbl_Parameter) in the database for the team project collection. Inspecting the UserAgent column in tbl_Command lets TFS administrators figure out the source of requests.

Figure 13 Determining the First E-mail in a Thread

```
// Get either the folder or item identifier for a
// create/delete/new mail event.
if (bocet.Item is ItemIdType)
{
    // Get the item identifier.
    ItemIdType itemId = bocet.Item as ItemIdType;

    //

    // Get the message using the ItemIdType.
    //

    // From web.config
    string ewsServiceUrl =
        ConfigurationManager.AppSettings["EWSServiceUrl"];

    ExchangeService esb =
        new ExchangeService(ExchangeVersion.Exchange2010);
    esb.Url = new Uri(ewsServiceUrl);
    // From Web service App Pool identity.
    esb.UseDefaultCredentials = true;

    EmailMessage message = EmailMessage.Bind(esb, new ItemId(itemId.Id));
    ConvertEmailToRequestItem(message);
}
```

TEAM FOUNDATION SERVER(TFS) HOSTING

Hosted TFS SaaS Solution for Source Control and Bug/Issue Tracking

discount
Asp.net
Team Foundation Server Hosting

*Source Code Version Control
Bug/Issue Tracking
Unlimited Projects
Visual Studio 2010/2008 Integration
5gb of Disk Space
USA & European Data Centers
Monthly Billing
Multi-User Discounts*



Microsoft® Partner
Gold Hosting

*No Risk 30 Day
FREE Trial!*

*No
Setup Fees*

www.DiscountASP.NET/tfs/msdn

*FREE VSS to
Hosted TFS
Migration Services*

*NEW!
TFS BUILD
SERVER Add-on*



The notification Web service customizes the user agent data in order to clearly identify it by using the following statement:

```
TfsConnection.ApplicationName = string.Format(
    TFS_APPLICATIONNAME_TEMPLATE, WindowsIdentity.GetCurrent().Name);
```

In this particular case, the user agent will look something like "MsdnMagSubscribeSvc - Contoso\JDoe."

Another potentially useful feature of ConvertEmailToRequestItem is its ability to filter e-mails according to a certain prefix in the subject line, as in the following code:

```
// First check if prefix-based filtering is turned on, and if so,
// if the e-mail subject line matches the defined prefix.
string subjectPrefix =
    ConfigurationManager.AppSettings["SubjectPrefix"];
if (!string.IsNullOrEmptyWhiteSpace(subjectPrefix))
{
    if (!title.StartsWith(subjectPrefix,
        StringComparison.InvariantCultureIgnoreCase))
    {
        WriteEventToLog(
            string.Format(SUBJECT_DOES_NOT_MATCH_PREFIX_TEMPLATE,
                subjectPrefix, title));
    }
}
return;
```

You can use this feature to restrict the service's functionality to only certain e-mails; for example, to code review e-mails where the prefix can be defined as "Code Review."

Other Use Cases: Code Reviews

The use case presented here is a support ticket. However, the flexibility of the approach presented in this article makes it easy to apply the same techniques to other use cases, such as tracking code reviews. The process flow for the code review use case is depicted in **Figure 14**.

Development organizations tend to conduct code reviews via e-mail. There have been many attempts to integrate code reviews into TFS by creating code review work items and building add-ins to allow managing the code review process from within Visual Studio. However, the same technique employed for support tickets can provide an easy way to integrate code review e-mail threads into the TFS work item tracking system. The following steps outline a suggested process for using the notifications infrastructure with code reviews:

- Start by creating a Code Review work item similar to the Support Request work item.
- Set up a watcher account in a DL to monitor e-mails sent to the DL.
- Use the SubjectPrefix feature presented earlier to define a prefix that distinguishes e-mails as code review e-mails. Code review requests sent via e-mail often have the subject line follow this pattern: "Code Review: My-Shelveset-Name."

**Work items can be linked
to changesets.**

As shown in **Figure 14**, associating the actual check-in of the shelveset contents with the work item provides traceability into the lifetime of the code from when the code was done, when it was sent for code review, when it was approved for check-in and, finally, to when it was checked in. This association is a powerful feature of

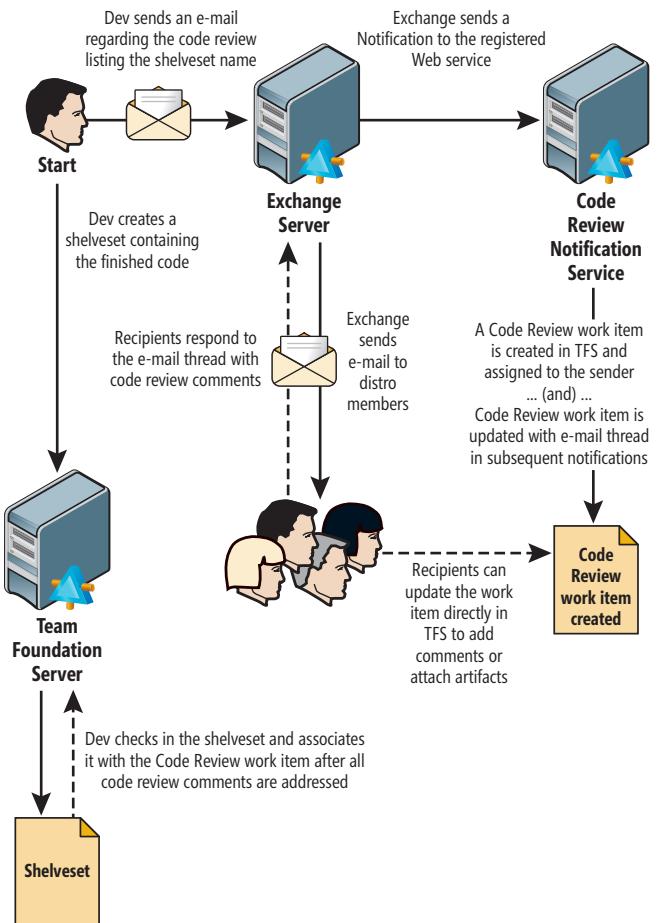


Figure 14 Process Flow for the Code Review Work Item

TFS that provides great visibility into any code that gets checked in via source control. It's an especially useful asset for organizations that require traceability for audit reasons.

What Next

The key goal of this article was to demonstrate one of the many possibilities for using TFS and Exchange together to increase both productivity and efficiency. One of the great things about work items, especially in TFS 2010, is how they're integrated inside TFS. Work items can be linked to changesets. They can represent and track test plans. They can be easily reported on, using Excel or SSRS, and they can also be synchronized with Microsoft Project.

Armed with the use case and techniques presented in this article, you should be able to leverage the notifications Web service and the work item customizations in many other scenarios that are particular to your own environment. ■

MOHAMMAD JALLOUL is a software engineer at Myspace Inc., a social entertainment Web site where he's responsible for integrating TFS into the different business processes and leveraging it to increase efficiency and productivity. Prior to that, he worked in Redmond in the Developer Division at Microsoft on an internal tool called Gibraltar. He can be reached through his blog at mohammadjalloul.com.

THANKS to the following technical expert for reviewing this article: Bill Heys

WORD PROCESSING COMPONENTS

MILES BEYOND RICH TEXT



- TRUE WYSIWYG
- POWERFUL MAIL MERGE
- MS OFFICE NOT REQUIRED
- PDF, DOCX, DOC, RTF & HTML



Word Processing Components
for Windows Forms & ASP.NET

WWW.TEXTCONTROL.COM



TX Text Control Sales:

US +1 877-462-4772 (toll-free)

EU +49 421-4270671-0

The Past, Present and Future of Parallelizing .NET Applications

Stephen Toub

The Ghost of Parallelism Past

Direct thread manipulation has historically been the way developers attempted to achieve responsive client applications, parallelized algorithms and scalable servers. Yet such techniques have also been the way developers historically achieved deadlocks, livelocks, lock convoys, two-step dances, race conditions, oversubscription and a host of other undesirable warts on applications. Since its inception, the Microsoft .NET Framework has provided a myriad of lower-level tools for building concurrent applications, including an entire namespace dedicated to the endeavor: System.Threading. With approximately 50 types in this namespace in the .NET Framework 3.5 core assemblies (including such types as Thread, ThreadPool, Timer, Monitor, ManualResetEvent, ReaderWriterLock and Interlocked), no one should be able to accuse the .NET Framework of being light on threading support. And yet, I will accuse previous versions of the .NET Framework as being light on the real support developers everywhere need to successfully build scalable and highly parallelized applications. This is a problem I'm thankful and excited to say has been rectified in the .NET Framework 4, and it's continuing to see a significant amount of investment for future .NET Framework versions.

This article discusses:

- Limitations of earlier versions of the .NET Framework for building concurrent applications
- Parallelization support in the .NET Framework 4
- The future for parallelism and concurrency in the .NET Framework

Technologies discussed:

Microsoft .NET Framework, TPL, Async, PLINQ

Some may question the value of a rich subsystem in a managed language for writing parallel code. After all, parallelism and concurrency are about performance, and developers interested in performance should seek out native languages that provide pedal-to-the-metal access to the hardware and full control over every bit twiddle, cache line manipulation and interlocked operation ... right? I fear for the state of our industry if that is indeed the case. Managed languages like C#, Visual Basic and F# exist to provide all developers—mere mortals and superheroes alike—with a safe, productive environment in which to rapidly develop powerful and efficient code. Developers are provided with thousands upon thousands of prebuilt library classes, along with languages ripe with all of the modern services we've come to expect, and still manage to achieve impressive performance numbers on all but the most number-crunching and floating-point-intensive workloads. All of this is to say that managed languages and their associated frameworks have deep-seated support for building high-performing concurrent applications, so that developers on modern hardware can have their cake and eat it too.

I've always felt that patterns are a great way to learn, so for the topic at hand it's only right that we start our exploration by looking at a pattern. For the "embarrassing parallel" or "delightfully parallel" pattern, one of the most commonly needed fork-join constructs is a parallel loop, which is intended to process every independent iteration of a loop in parallel. It's instructive to see how such processing could be done using the lower-level primitives mentioned earlier, and for that we'll walk through the basic implementation of a naïve parallel loop implemented in C#. Consider a typical for loop:

```
for (int i=0; i<N; i++) {  
    ... // Process i here  
}
```

We can use threads directly to effect the parallelization of this loop, as shown in **Figure 1**.

Of course, there are a myriad of problems with this parallelization approach. We're spinning up new threads dedicated to the loop, which not only add overhead (in particular if the body of the loop is trivial in work to be done) but can also lead to significant oversubscription in a process that's doing other work concurrently. We're using static partitioning to divide the work among the threads, which could lead to significant load imbalance if the workload is not evenly distributed across the iteration space (not to mention that if the number of iterations isn't evenly divided by the number of utilized threads, the last thread is burdened with the overflow). Arguably worst of all, however, is that the developer is forced to write this code in the first place. Every algorithm we attempt to parallelize will require similar code—code that's brittle at best.

The problem exemplified by the previous code is further amplified when we recognize that parallel loops are just one pattern in the multitudes that exist in parallel programs. Forcing developers to express all such parallel patterns at this low level of coding does not make for a good programming model, and does not set up for success the masses of developers in the world who need to be able to utilize massively parallel hardware.

The Ghost of Parallelism Present

Enter the .NET Framework 4. This release of the .NET Framework was augmented with a multitude of features to make it significantly easier for developers to express parallelism in their applications, and to have that parallelism executed efficiently. This goes well beyond parallel loops, but we'll begin there nonetheless.

The System.Threading namespace was enhanced in the .NET Framework 4 with a new sub namespace: System.Threading.Tasks. This namespace includes a new type, Parallel, that exposes an abundance of static methods for implementing parallel loops and structured fork-join patterns. As an example of its usage, consider the previous for loop:

```
for (int i=0; i<N; i++) {
    ... // Process i here
}
```

With the Parallel class, you can parallelize it simply as follows:

```
Parallel.For(0, N, i => {
    ... // Process i here
});
```

Here, the developer is still responsible for ensuring each iteration of the loop is in fact independent, but beyond that, the Parallel.For construct handles all aspects of this loop's parallelization. It handles dynamically partitioning the input range across all underlying threads involved in the computation, while still minimizing overhead for partitioning close to those provided by static partitioning implementations. It handles scaling up and scaling down the number of threads involved in the computation dynamically in order to find the optimal number of threads for a given workload (which is not always equal to the number of hardware threads, contrary to popular belief). It provides exception-handling capabilities not present in my naïve implementation previously shown, and so on. Most importantly, it keeps the developer from having to think about parallelism at the lower-level OS abstraction of threads, and from needing to continually code delicate solutions for partitioning workloads, offloading to multiple cores and joining the results efficiently. Instead, it enables the developer to spend his time focusing on what's important: the business logic that makes the developer's work profitable.

Figure 1 Parallelizing a For Loop

```
int lowerBound = 0, upperBound = N;
int numThreads = Environment.ProcessorCount;
int chunkSize = (upperBound - lowerBound) / numThreads;

var threads = new Thread[numThreads];
for (int t = 0; t < threads.Length; t++) {
    int start = (chunkSize * t) + lowerBound;
    int end = t < threads.Length - 1 ? start + chunkSize : upperBound;
    threads[t] = new Thread(delegate() {
        for (int i = start; i < end; i++) {
            ... // Process i here
        }
    });
}

foreach (Thread t in threads) t.Start(); // fork
foreach (Thread t in threads) t.Join(); // join
```

Parallel.For also provides facilities for developers who require more fine-grained control over a loop's operation. Through an options bag provided to the For method, developers can control the underlying scheduler the loop runs on, the maximum degree of parallelism to be employed, and the cancellation token used by an entity external to the loop to request the loop's polite termination at the loop's earliest convenience:

```
var options = new ParallelOptions { MaxDegreeOfParallelism = 4 };
Parallel.For(0, N, options, i => {
    ... // Process i here
});
```

This customization capability highlights one of the goals of this parallelization effort within the .NET Framework: To make it significantly easier for developers to take advantage of parallelism without complicating the programming, but at the same time give more advanced developers the knobs they need to fine-tune the processing and execution. In this vein, additional tweaks are supported. Other overloads of Parallel.For enable developers to break out of the loop early:

```
Parallel.For(0, N, (i,loop) => {
    ... // Process i here
    if (SomeCondition()) loop.Break();
});
```

Still other overloads allow developers to flow state through iterations that end up running on the same underlying thread, enabling far more efficient implementations of algorithms such as reductions, for example:

```
static int SumComputations(int [] inputs, Func<int,int> computeFunc) {
    int total = 0;
    Parallel.For(0, inputs.Length, () => 0, (i,loop,partial)=> {
        return partial + computeFunc(inputs[i]);
    },
    partial => Interlocked.Add(ref total, partial));
}
```

The Parallel class provides support not just for integral ranges, but also for arbitrary IEnumerable<T> sources, the .NET Framework representation of an enumerable sequence: code may continually call MoveNext on an enumerator in order to retrieve the next Current value. This ability to consume arbitrary enumerables enables parallel processing of arbitrary data sets, regardless of their in-memory representation; data sources may even be materialized on demand, and paged in as MoveNext calls reach not-yet-materialized sections of the source data:

```
IEnumerable<string> lines = File.ReadLines("data.txt");
Parallel.ForEach(lines, line => {
    ... // Process line here
});
```

As is the case with Parallel.For, Parallel.ForEach sports a multitude of customization capabilities, offering greater control than

Parallel.For. For example, ForEach lets a developer customize how the input data set is partitioned. This is done through a set of partitioning-focused abstract classes that enable parallelization constructs to request a fixed or variable number of partitions, allowing the partitioner to hand out those partition abstractions over the input data set and to assign data to those partitions statically or dynamically as appropriate:

```
Graph<T> graph = ...;
Partitioner<T> data = new GraphPartitioner<T>(graph);
Parallel.ForEach(data, vertex => {
    ... // Process vertex here
});
```

Parallel.For and Parallel.ForEach are complemented on the Parallel class by an Invoke method that accepts an arbitrary number of actions to be invoked with as much parallelism as the underlying system can muster. This classic fork-join construct makes it easy to parallelize recursive divide-and-conquer algorithms, such as the commonly used example of QuickSort:

```
static void QuickSort<T>(T [] data, int lower, int upper) {
    if (upper - lower < THRESHOLD) {
        Array.Sort(data, index:lower, length:upper-lower);
    }
    else {
        int pivotPos = Partition(data, lower, upper);
        Parallel.Invoke(
            () => QuickSort(data, lower, pivotPos),
            () => QuickSort(data, pivotPos, upper));
    }
}
```

While a great step forward, the Parallel class only scratches the surface of the functionality available. One of the more monumental parallelization strides taken in the .NET Framework 4 was the introduction of Parallel LINQ, lovingly referred to as PLINQ (pronounced “Pee-link”). LINQ, or Language Integrated Query, was introduced to the .NET Framework in version 3.5. LINQ is really two things: a description of a set of operators exposed as methods for manipulating data sets, and contextual keywords in both C# and Visual Basic for expressing these queries directly in language. Many of the operators included in LINQ are based on the equivalent operations known for years to the database community, including Select, SelectMany, Where, Join, GroupBy and approximately 50 others. The .NET Framework Standard Query Operators API defines the pattern for these methods, but it doesn’t define which exact data sets these operations should target, nor exactly how these operations should be implemented. Various “LINQ providers” then implement this pattern for a multitude of different data sources and target environments (in-memory collections, SQL databases, object/relational mapping systems, HPC Server compute clusters, temporal and streaming data sources and more). One of the most commonly used providers is called LINQ to Objects, and it provides the full suite of LINQ operators implemented on top of `IEnumerable<T>`. This enables the implementation of queries in C# and Visual Basic, such as the following snippet that reads all data from a file line by line, filtering down to just those lines that contain the word “secret” and encrypting them. The end result is an enumerable of byte arrays:

```
IEnumerable<byte[]> encryptedLines =
    from line in File.ReadLines("data.txt")
    where line.Contains("secret")
    select DataEncryptor.Encrypt(line);
```

For computationally intensive queries, or even just for queries that involve a lot of long-latency I/O, PLINQ provides auto-

matic parallelization capabilities, implementing the full LINQ operator set utilizing end-to-end parallel algorithms. Thus, the previous query can be parallelized simply by the developer appending `.AsParallel()` to the data source:

```
IEnumerable<byte[]> encryptedLines =
    from line in File.ReadLines("data.txt").AsParallel()
    where line.Contains("secret")
    select DataEncryptor.Encrypt(line);
```

As with the Parallel class, this model is opt-in to force the developer to evaluate the ramifications of running the computation in parallel. Once that choice has been made, however, the system handles the lower-level details of the actual parallelization, partitioning, thread throttling and the like. Also, as with Parallel, these PLINQ queries are customizable in a variety of ways. A developer can control how partitioning is achieved, how much parallelism is actually employed, tradeoffs between synchronization and latency, and more:

```
IEnumerable<byte[]> encryptedLines =
    from line in new OneAtATimePartitioner<string>(
        File.ReadLines("data.txt"))
    .AsParallel()
    .AsOrdered()
    .WithCancellation(someExternalToken)
    .WithDegreeOfParallelism(4)
    .WithMergeOptions(ParallelMergeOptions.NotBuffered)
    where line.Contains("secret")
    select DataEncryptor.Encrypt(line);
```

These powerful and higher-level programming models for loops and queries are built on top of an equally powerful but lower-level set of task-based APIs, centering around the Task and `Task<TResult>` types in the System.Threading.Tasks namespace. In effect, the parallel loops and query engines are task generators, relying on the underlying task infrastructure to map the parallelism expressed to the resources available in the underlying system. At its core, Task is a representation for a unit of work—or, more generally, for a unit of asynchrony, a work item that may be spawned and later joined with through various means. Task provides `Wait`, `WaitAll` and `WaitAny` methods that allow for synchronously blocking forward progress until the target task (or tasks) has completed, or until additional constraints supplied to overloads of these methods have been met (for instance, a timeout or cancellation token). Task supports polling for completion through its `IsCompleted` property, and, more generally, polling for changes in its lifecycle processing through its `Status` property. Arguably most importantly, it provides the `ContinueWith`, `ContinueWhenAll` and `ContinueWhenAny` methods, which enable the creation of tasks that will be scheduled only when a specific set of antecedent tasks have completed. This continuation support allows a myriad of scenarios to be implemented easily, enabling dependencies to be expressed between computations such that the system can schedule work based on those dependencies becoming satisfied:

```
Task t1 = Task.Factory.StartNew(() => BuildProject(1));
Task t2 = Task.Factory.StartNew(() => BuildProject(2));
Task t3 = Task.Factory.StartNew(() => BuildProject(3));
Task t4 = Task.Factory.ContinueWhenAll(
    new [] { t1, t2 }, _ => BuildProject(4));
Task t5 = Task.Factory.ContinueWhenAll(
    new [] { t2, t3 }, _ => BuildProject(5));
Task t6 = Task.Factory.ContinueWhenAll(
    new [] { t4, t5 }, _ => BuildProject(6));
t6.ContinueWith(_ => Console.WriteLine("Solution build completed."));
```

The `Task<TResult>` class derived from Task enables results to be passed out from the completed operation, providing the .NET Framework with a core “future” implementation:

```

int SumTree<T>(Node<T> root, Func<T,int> computeFunc) {
    if (root == null) return 0;
    Task<int> left = Task.Factory.StartNew(() => SumTree(root.Left));
    Task<int> right = Task.Factory.StartNew(() => SumTree(root.Right));
    return computeFunc(root.Data) + left.Result + right.Result;
}

```

Under all of these models (loops, queries and tasks alike) the .NET Framework employs work-stealing techniques to provide for more efficient processing of specialized workloads, and by default it employs hill-climbing heuristics to vary the number of employed threads over time in order to find the optimal processing level. Heuristics are also built into pieces of these components to automatically fall back to sequential processing if the system believes that any parallelization attempt would result in slower-than-sequential result times—though, as with the other defaults discussed previously, these heuristics may also be overridden.

Task<TResult> need not represent only compute-bound operations. It may also be used to represent arbitrary asynchronous operations. Consider the .NET Framework System.IO.Stream class, which provides a Read method to extract data from the stream:

```

NetworkStream source = ...;
byte [] buffer = new byte[0x1000];
int numBytesRead = source.Read(buffer, 0, buffer.Length);

```

This Read operation is synchronous and blocking, such that the thread making the Read call may not be used for other work until the I/O-based Read operation completes. To allow better scalability, the Stream class provides an asynchronous counterpart to the Read method in the form of two methods: BeginRead and EndRead. These methods follow a pattern available in the .NET Framework since its inception, a pattern known as the APM, or Asynchronous Programming Model. The following is an asynchronous version of the previous code example:

```

NetworkStream source = ...;
byte [] buffer = new byte[0x1000];
source.BeginRead(buffer, 0, buffer.Length, delegate(IAsyncResult iar) {
    int numBytesRead = source.EndRead(iar);
}, null);

```

This approach, however, leads to poor composability. The TaskCompletionSource<TResult> type fixes this by enabling such an asynchronous read operation to be exposed as a task:

```

public static Task<int> ReadAsync(
    this Stream source, byte [] buffer, int offset, int count)
{
    var tcs = new TaskCompletionSource<int>();
    source.BeginRead(buffer, 0, buffer.Length, iar => {
        try { tcs.SetResult(source.EndRead(iar)); }
        catch(Exception exc) { tcs.SetException(exc); }
    }, null);
    return tcs.Task;
}

```

This allows for multiple asynchronous operations to be composed just as in the compute-bound examples. The following example concurrently reads from all of the source Streams, writing out to the console only when all of the operations have completed:

```

NetworkStream [] sources = ...;
byte [] buffers = ...;
Task.Factory.ContinueWhenAll(
    (from i in Enumerable.Range(0, sources.Length)
     select sources[i].ReadAsync(buffers[i], 0, buffers[i].Length))
    .ToArray(),
    _ => Console.WriteLine("All reads completed"));

```

Beyond mechanisms for launching parallelized and concurrent processing, the .NET Framework 4 also provides primitives for further coordinating work between tasks and threads. This includes a set of thread-safe and scalable collection types that largely eliminate the

need for developers to manually synchronize access to shared collections. ConcurrentQueue<T> provides a thread-safe, lock-free, first-in-first-out collection that may be used concurrently by any number of producers and any number of consumers. In addition, it supports snapshot semantics for concurrent enumerators, so that code may examine the state of the queue at a moment in time even as other threads bash away at the instance. ConcurrentStack<T> is similar, instead providing last-in-first-out semantics. ConcurrentDictionary<T> uses lock-free and fine-grained locking techniques to provide a thread-safe dictionary, which also supports any number of concurrent readers, writers and enumerators. It also provides several atomic implementations of multistep operations, such as GetOrAdd and AddOrUpdate. Another type, ConcurrentBag<T>, provides an unordered collection using work-stealing queues.

The .NET Framework doesn't stop at collection types. Lazy<T> provides lazy initialization of a variable, using configurable approaches to achieve thread safety. ThreadLocal<T> provides per-thread, per-instance data that can also be lazily initialized on first access. The Barrier type enables phased operation so that multiple tasks or threads can proceed through an algorithm in lockstep. The list continues, and all stem from a single guiding principle: Developers shouldn't need to focus on the lower-level and rudimentary aspects of their algorithm's parallelization, and instead allow the .NET Framework to handle the mechanics and efficiency details for them.

The Ghost of Parallelism Yet to Come

Unlike the Dickens counterpart, the future for parallelism and concurrency in the .NET Framework is exciting and something to look forward to, building upon the foundations laid in the .NET Framework 4. A focus of future versions of the .NET Framework, beyond improving the performance of the existing programming models, is expanding the set of higher-level models that exist in order to address more patterns of parallel workloads. One such enhancement is a new library for implementing parallel systems based on dataflow and for architecting applications with agent-based models. The new System.Threading.Tasks.Dataflow library provides a multitude of "dataflow blocks" that act as buffers, processors and propagators of data. Data may be posted to these blocks, and that

Figure 2 Using a BlockingCollection

```

static BlockingCollection<Datum> s_data =
    new BlockingCollection<Datum>(boundedCapacity:100);
...
static void Producer() {
    for(int i=0; i<N; i++) {
        Datum d = GenerateData();
        s_data.Add(d);
    }
    s_data.CompleteAdding();
}

static void Consumer() {
    foreach(Datum d in s_data.GetConsumingEnumerable()) {
        Process(d);
    }
}
...
var workers = new Task[3];
workers[0] = Task.Factory.StartNew(Producer);
workers[1] = Task.Factory.StartNew(Consumer);
workers[2] = Task.Factory.StartNew(Consumer);
Task.WaitAll(workers);

```

data will be processed and automatically forwarded to any linked targets, based on the semantics of the source block. The dataflow library is also built on top of tasks, with the blocks spinning up tasks under the covers to process and propagate data.

From a patterns perspective, the library is particularly good at handling dataflow networks that form chains of producers and consumers. Consider the need for data to be compressed and then encrypted and written out to a file, with the stream of data arriving and flowing through the application. This might be achieved by configuring a small network of dataflow blocks as follows:

```
static byte[] Compress(byte[] data) { ... }
static byte[] Encrypt(byte[] data) { ... }

...
var compressor = new TransformBlock<byte[], byte[]>(Compress);
var encryptor = new TransformBlock<byte[], byte[]>(Encrypt);
var saver = new ActionBlock<byte[]>(AppendToFile);
compressor.LinkTo(encryptor);
encryptor.LinkTo(saver);

...
// As data arrives
compressor.Post(byteArray);
```

Beyond the dataflow library, however, arguably the most important feature coming for parallelism and concurrency in the .NET Framework is first-class language support in C# and Visual Basic for producing and asynchronously awaiting tasks. These languages are being augmented with state-machine-based rewrite capabilities that allow for all of the languages' sequential control flow constructs to be utilized while at the same time being able to asynchronously wait for tasks to complete (F# in Visual Studio 2010 supports a related form of asynchrony as part of its asynchronous workflows feature, a feature that also integrates with tasks). Take a look at the following method, which synchronously copies data from one Stream to another, returning the number of bytes copied:

```
static long CopyStreamToStream(Stream src, Stream dst) {
    long numCopied = 0;
    byte[] buffer = new byte[0x1000];
    int numRead;
    while((numRead = src.Read(buffer, 0, buffer.Length)) > 0) {
        dst.Write(buffer, 0, numRead);
        numCopied += numRead;
    }
    return numCopied;
}
```

Implementing this function, including its conditionals and loops, with support like the BeginRead/EndRead methods on Stream shown earlier, results in a nightmare of callbacks and logic that's error-prone and extremely difficult to debug. Instead, consider the approach of using a ReadAsync method like the one shown earlier, which returns a Task<int>, and a corresponding WriteAsync method, which returns a Task. Using the new C# functionality, we can rewrite the previous method as follows:

```
static async Task<long> CopyStreamToStreamAsync(Stream src, Stream dst) {
    long numCopied = 0;
    byte[] buffer = new byte[0x1000];
    int numRead;
    while((numRead = await src.ReadAsync(buffer, 0, buffer.Length)) > 0) {
        await dst.WriteAsync(buffer, 0, numRead);
        numCopied += numRead;
    }
    return numCopied;
}
```

Notice the few minor modifications to turn the synchronous method into an asynchronous method. The function is now annotated as "async" to inform the compiler that it should perform a rewrite of the function. With that, any time an "await" operation is requested on a Task or Task<TResult>, the rest of the functions'

execution is, in effect, hooked up to that task as a continuation: until such time that the task completes, this method call won't be occupying a thread. The Read method call has been converted into a ReadAsync call, such that the await contextual keyword may be used to signify the yielding point where the remainder should be turned into a continuation; it's the same for WriteAsync. When this asynchronous method eventually completes, the returned long value will be lifted into a Task<long> that was returned to the initial caller of CopyStreamToStreamAsync, using a mechanism like the one shown earlier with TaskCompletionSource<TResult>. I can now use the return value from CopyStreamToStreamAsync as I would any Task, waiting on it, hooking up a continuation to it, composing with it over other tasks or even awaiting it. With functionality like ContinueWhenAll and WaitAll, I can initiate and later join with multiple asynchronous operations in order to achieve higher levels of concurrency and improve the overall throughput of my application.

This language support for asynchrony improves not only I/O-bound but also CPU-bound operations, and in particular the ability of a developer to build responsive client applications (ones that don't tie up the UI thread and leave the application in a non-responsive state) while still getting the benefits of massively parallel processing. It has long been cumbersome for developers to move off of a UI thread, perform any processing, and then move back to the UI thread to update UI elements and interact with a user. The language support for asynchrony interacts with key components of the .NET Framework to, by default, automatically bring operations back to their original context when an await operation completes (for instance, if an await is issued from the UI thread, the continuation that's hooked up will continue execution back on the UI thread). This means that a task can be launched to run compute-intensive work in the background, and the developer can simply await it to retrieve the results and store them into UI elements, like so:

```
async void button1_Click(object sender, EventArgs e) {
    string filePath = txtFilePath.Text;
    txtOutput.Text = await Task.Factory.StartNew(() => {
        return ProcessFile(filePath);
    });
}
```

Figure 3 Using a BufferBlock

```
static BufferBlock<Datum> s_data = new BufferBlock<Datum>(
    new DataflowBlockOptions { BoundedCapacity=100 });
...
static async Task ProducerAsync() {
    for(int i=0; i<N; i++) {
        Datum d = GenerateData();
        await s_data.SendAsync(d);
    }
    s_data.Complete();
}

static async Task ConsumerAsync() {
    Datum d;
    while(await s_data.OutputAvailableAsync()) {
        while(s_data.TryReceive(out d)) {
            Process(d);
        }
    }
}
...
var workers = new Task[3];
workers[0] = ProducerAsync();
workers[1] = ConsumerAsync();
workers[2] = ConsumerAsync();
await Task.WhenAll(workers);
```

That background task may itself spin off multiple tasks in order to parallelize the background computation, such as by using a PLINQ query:

```
async void button1_Click(object sender, EventArgs e) {
    string filePath = txtFilePath.Text;
    txtOutput.Text = await Task.Factory.StartNew(() => {
        return File.ReadLines(filePath).AsParallel()
            .SelectMany(line => ParseWords(line))
            .Distinct()
            .Count()
            .ToString();
    });
}
```

The language support may also be used in combination with the dataflow library to ease the natural expression of asynchronous producer/consumer scenarios. Consider the desire to implement a set of throttled producers, each of which is generating some data to be sent off to a number of consumers. Synchronously, this might be done using a type like BlockingCollection<T> (see **Figure 2**), which was introduced as part of the .NET Framework 4.

This is a fine pattern, as long as it meets the application's goals that both the producers and consumers block threads. If that's unacceptable, you can write an asynchronous counterpart, utilizing another of the dataflow blocks, BufferBlock<T>, and the ability to asynchronously send to and receive from a block, as shown in **Figure 3**.

Here, the SendAsync and OutputAvailableAsync methods both return tasks, enabling the compiler to hook up continuations and allowing the whole process to run asynchronously.

A Scrooge No More

Parallel programming has long been the domain of expert developers, uniquely qualified individuals well-versed in the art of scaling code to multiple cores. These experts are the result of years of training and on-the-job experiences. They're highly valued—and they're scarce. In our brave new world of multicore and manycore everywhere, this model of leaving parallelism purely to the experts is no longer sufficient. Regardless of whether an application or component is destined to be a publicly available software package, is intended purely for in-house use, or is just a tool to enable a more important job to be completed, parallelism is now something every developer must at least consider, and something that the millions upon millions of developers that utilize managed languages must be able to take advantage of,

even if it's through components that themselves encapsulate parallelism. Parallel programming models like those exposed in the .NET Framework 4, and like those coming in future versions of the .NET Framework, are necessary to enable this beautiful future. ■

STEPHEN TOUB is a principal architect on the Parallel Computing Platform team at Microsoft.

THANKS to the following technical experts for reviewing this article:
Joe Hoag and Danny Shih

Less Plumbing Code, More Features use CODEFLUENT ENTITIES!

Focus on what makes the difference

Define your business logic, choose your technical targets, and create your custom business rules and behaviors!

Your application deserves rock-solid foundations: let CodeFluent Entities generate them, and keep yourself the fun part!

CodeFluent Entities provides a Visual Studio 2008/2010 integrated environment that helps you master present and future Microsoft .NET development technologies.



CodeFluent Entities is a model-first tool for continuous generation of all your application layers (user interface, service, business and database) that preserves your custom code. It offers a framework-free, UML-free, ORM-free, and template-free approach. Key features include:

- ✓ FRAMEWORK FREE
- ✓ UML FREE
- ✓ ORM FREE
- ✓ TEMPLATE FREE



A model-first tool for continuous generation of all your application layers (user interface, service, business and database) that preserves your custom code.

**DOWNLOAD YOUR LICENSE
TOTALLY FREE FOR PERSONAL USAGE**

www.codefluententities.com/msdn

**CodeFluent
Entities**
the Model Driven Software factOry

Contact: info@softfluent.com
Twitter: twitter.com/softfluent
US Sales: +1 425 372 3047
Europe Sales: +33 1 75 60 04 45

CodeFluent Entities is a trademark of SoftFluent SAS.
All other product and brand names are trademarks and/or registered trademarks of their respective holders

Portable Class Libraries: A Primer

Bill Kratochvil

A Portable Class Library (PCL) project will generate a managed assembly that can be referenced by Windows Phone 7, Silverlight, the Microsoft .NET Framework and Xbox 360 platforms. This can maximize reuse of your code and reduce the number of required projects, particularly in multi-targeted applications that share the same codebase, as is the case of the demo application that accompanies this article. I invested time in writing a Windows Phone 7 application for this article, and the WPF/Silverlight applications came for free. The only constraint is that the PCL can't reference platform-specific projects; it can only reference other PCL projects. On the surface this can appear limiting, especially for applications that utilize Prism and dependency injection (DI). But with careful planning, you can work around this

This article discusses:

- A multi-targeted password manager application
- Using dependency injection frameworks
- Use cases

Technologies discussed:

Portable Class Libraries, Windows Phone 7, Silverlight, Microsoft .NET Framework

Code download available at:

code.msdn.microsoft.com/mag201108PCL

constraint and create efficient PCL projects that will help enforce good development practices.

Prior to PCL projects, solution projects could only reference assemblies of the same platform. Silverlight projects referenced other Silverlight assemblies, .NET projects other .NET assemblies and so on. To code effectively, we could accumulate an unmanageable number of projects; when creating shared codebases (code that can be used across all platforms), we'd have to create a project for each platform.

A Multi-Targeted Password Manager Solution

The Password Manager solution (passwordmgr.codeplex.com) is a multi-targeted application with a single codebase (hosted by the Windows Phone 7 projects with the Silverlight and WPF projects linking to the Phone project files). As you can see, this small application has a daunting number of projects.

Why so many projects? That's a popular question that I often hear from developers as I architect frameworks for client solutions. A typical refrain is, "They make the solution confusing and difficult to understand," which is a good argument.

My answer is always: "To have a clear separation of concerns and to maximize reuse." Each project should have one purpose (and do it well in a loosely coupled manner). As a case in point, note the apparent separation available in the PasswordMgr solution and the ability to easily substitute a different data access layer (DAL) using DI. Nevertheless, you'd be forced to pull in the SQLite assemblies and associated projects to reuse this code, even if you didn't plan

on using SQLite (perhaps you'd be using SQL Server for your DAL).

Once you reference another project, your project can become tightly coupled, forcing you to drag not only it, but also any dependencies it has, to the other projects. If you only have a few projects, it makes it increasingly more difficult to reuse the code in other solutions.

A PCL can significantly reduce the number of projects you have to manage, particularly if you want to have a clear separation of concerns permitting you to easily reuse your projects in other modules or solutions. The key is to keep your projects loosely coupled by programming against interfaces. This will permit you to use DI frameworks, such as the Managed Extensibility Framework (MEF) and Unity, which allow you to easily configure the implementation for the interfaces. That is, the DAL implementation for interfaces could be SQL Server, the cloud or SQLite classes.

Once you install the prerequisites (as outlined in the MSDN documentation at bit.ly/fxatk0), you'll have a new feature under "Add New Project" to create a PCL.

Using DI Frameworks

A question that will quickly arise as you attempt to use these powerful DI frameworks is, "How do I use the PCL with these frameworks if I can't reference the DI framework components—that is, the [Dependency] or [Export] attributes?" For example, the SecurityViewModel in the following code has a Unity [Dependency] attribute, which will resolve the implementation of the ISecurityViewModel:

```
namespace MsdnDemo.MvpVmViewModels
{
    public class SecurityViewModel : PresentationViewModelBase
    {
        [Dependency]
        public ISecurityViewModel UserInfo { get; set; }

        public bool IsAuthenticated
        {
            get { return UserInfo.IsAuthenticated; }
            set
            {
                UserInfo.IsAuthenticated = value;
                OnPropertyChanged("IsAuthenticated");

                if(value)
                {
                    IsAdmin = UserInfo.IsInRole("Admin");
                    IsGuest = UserInfo.IsInRole("Guest");
                }
            }
        }
    }
}
```

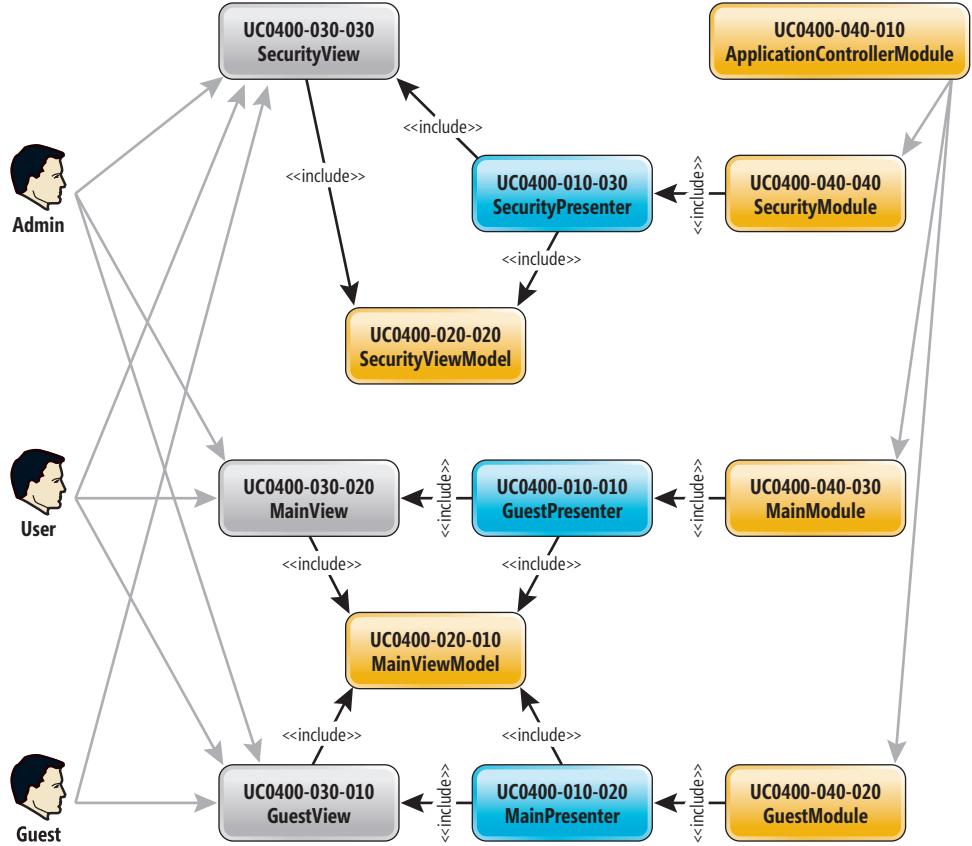


Figure 1 Tightly Coupled Components

Because you can only reference other PCL projects, it may seem impractical to use a PCL with DI or other shared resources that you have in your reusable libraries. In reality, this constraint can actually help enforce a clear separation of concerns, making your projects even more reusable—you may have projects in the future that won't utilize DI but could benefit from your PCL project.

I'll demonstrate the principles involved in using the PCL with DI using a demo application. It's a simple application that has two modules (guest and main) that are loaded on-demand using role-based security. Available features for these modules and Views will be determined by the roles attached to both the module and logged-in user. There are three users: Admin, Guest and Jane Doe (user). A business rule is that the Guest account can never access the main module.

Prior to PCL projects, solution projects could only reference assemblies of the same platform.

The magic of this application lies in its simplicity; you'll find no codebehind in the Views or domain objects polluted by UI requirements. The ViewModels hold UI-specific state and the Presenters manage the Views/ViewModels associated with their concerns.

UC000-020 Layers

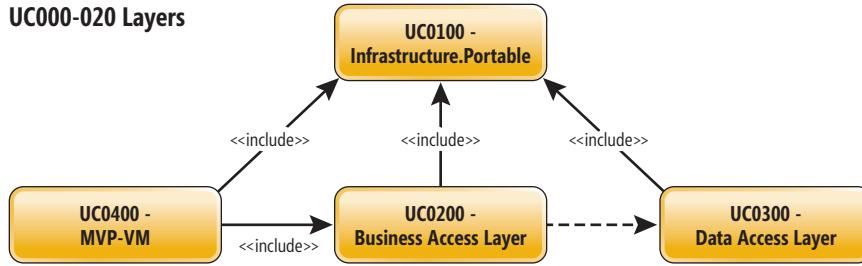


Figure 2 All Layers and Shared PCL Resources

through the use of business logic and DALs. Plumbing is handled by the infrastructure, such as button clicks, freeing up the developer to focus on business logic. Developers new to an application will quickly learn where to begin locating code—always start with the Presenter.

A PCL can significantly reduce the number of projects you have to manage.

Tightly Coupled Components

The only tight coupling is shown in **Figure 1**. This coupling is to be expected with the Model-View-Presenter ViewModel (MVPVM) pattern using Prism and DI. With this pattern, the module is responsible for instantiating the Presenter, which in turn instantiates the required View and ViewModel, wiring them up as required. The modules and their components have no knowledge of one another.

Ideally I would create separate projects for each module (ApplicationController, Main and Guest) so that I could reuse these modules (with this framework) in other solutions. (Note: Because both the GuestPresenter and MainPresenter share the MainViewModel, I'd also have to move this ViewModel into a shared project that could be accessed by both of them. You can see how quickly I can accumulate projects in the name of separating concerns and reusability, particularly if I were coding for multiple platforms.) You'll also see that, because I kept my demo simple, the only reusability it offers is copy and paste. The key is finding the balance, and the PCL helps provide that.

Figure 2 demonstrates how all layers (presentation, business and data) share PCL resources. Because security is a concern with nearly any application, I can safely integrate it—with its entities—

UC0100 - Infrastructure.Portable Overview

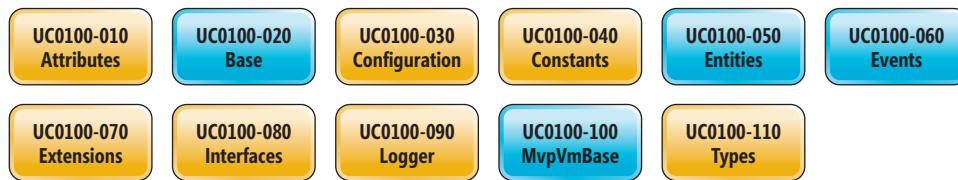


Figure 3 PCL Use Cases

in UC0100, which will be my PCL, with the notion that it may be the only reusable component that this demo application has (and a useful one at that).

It's beyond the scope of this article to cover all aspects of the PCL that I'll be using to manage my infrastructure; however, I'll cover the use cases highlighted in blue in **Figure 3**.

UC0100-020 Base Use Case

To improve extensibility in enterprise applications, it's helpful to have a consistent standard for wiring up the platform. This will help new developers as well as experienced developers who may not have frequented a module for a while. They'll be able to ramp up quickly to get required tasks done; minimal time will be spent hunting down code. With this in mind, I created a ModuleBase that's designed to work with the Prism IModule interface—more specifically, with the Initialize method. The issue here was that IModule wasn't available because it resides in a Prism (a non-PCL) assembly. I wanted the base to have logging capabilities, so the ILogger interface must be provided an instance before the Initialize method is called. This ModuleBase now can serve as a contract for all modules, in all solutions and projects, because it implements the IModule Initialize method, shown in **Figure 4**.

Unlike the PCL, my MsdnDemo.Phone project does have a platform-specific reference to my Prism assembly, so the DI code can reside in a PresentationModuleBase class within this project; it will be the base class for all modules. In a real-world application, this class, like other DI base classes, would reside in separate reusable projects.

To improve extensibility in enterprise applications, it's helpful to have a consistent standard for wiring up the platform.

When the module is resolved (instantiated) by the DI container, it will set Logger as configured by my GwnBootstrapper. When the Logger property value is set, it will pass the instance to the base

Logger, effectively providing the ModuleBase ILogger reference for the Initialize (and other) virtual methods (see **Figure 5**).

(Note: Because constructor injection occurs before setter injection—in the Unity framework—the constructor of ModuleBase can't have logging statements, because it will be null.)

Figure 4 The Initialize Method in the ModuleBase Class

```
public class ModuleBase
{
    public ILogger Logger { get; set; }

    /// <summary>
    /// Called by Prism catalog manager. Provides hook
    /// to register types/views and initialize the view model.
    /// </summary>
    public virtual void Initialize()
    {
        try
        {
            // Provide hooks for registrations
            RegisterTypes();
            RegisterViews();

            InitializeModule();
        }
        catch (Exception ex)
        {
            Logger.Log("ERROR in [{0}] {1}{2}",
                GetType().Name, ex.Message, ex.StackTrace);
        }
    }
}
```

Derived from PresentationModuleBase, the following MainModule code is the same code/file for all three platforms (Windows Phone 7, Silverlight and WPF):

```
public class MainModule : PresentationModuleBase
{
    protected override void RegisterViews()
    {
        base.RegisterViews();

        // Instantiate the presenter, which in turn will instantiate
        // (resolve) the View and ViewModel
        var presenter = Container.Resolve<MainPresenter>();

        // Load this presenters view into the MainRegion
        RegionViewRegistry
            .RegisterViewWithRegion(MvpVm.MainRegion, () => presenter.View);

        // Activate the presenters view after module is loaded
        RaiseViewEvent(MvpVm.MainModule, presenter.View.GetType().Name,
            ProcessType.ActivateView);
    }
}
```

UC0200-050 Entities Use Case

Using Plain Old CLR Objects (POCOs) provides the best reusability. Even though the PCL supports INotifyPropertyChanged, I may not always be using these entities with XAML. I may want to also use them with an ASP.NET MVC 3 project or the Entity

Figure 5 Setting the Logger Property

```
public class PresentationModuleBase : ModuleBase, IModule
{
    [Dependency]
    public override ILogger Logger { get; set; }

    [Dependency]
    public IUnityContainer Container { get; set; }

    [Dependency]
    public IRegionManager RegionManager { get; set; }

    [Dependency]
    public IEventAggregator EventAggregator { get; set; }

    [Dependency]
    public IRegionViewRegistry RegionViewRegistry { get; set; }
```

Framework. So my UserEntity and SecurityEntity objects can be enterprise-level POCO classes that can be easily reused in any application on any platform, as shown in **Figure 6**.

Using Plain Old CLR Objects (POCOs) provides the best reusability.

If the UserEntity POCO is going to be used on a ViewModel, it will have to be wrapped. The ViewModel will raise the notifications and, under the hood, the data will be transferred to the POCO, as shown in the excerpt from the MainViewModel class in **Figure 7**.

Note that I only show the FirstName property, but all of the UserEntity properties have a comparable wrapper. If I update SelectedUser, it will raise the property changed notification for all of the UserEntity properties so that XAML will be notified to update the UI fields as applicable.

Figure 6 The UserEntity and SecurityEntity Objects

```
public class UserEntity
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string PrimaryEmail { get; set; }
    public string Password { get; set; }
    public override string ToString()
    {
        return string.Format("{0} {1} ({2})", FirstName, LastName, Password);
    }
}

public class SecurityEntity : ISecurityViewModel
{
    private string _login;
    public int Id { get; set; }
    public bool IsAuthenticated { get; set; }
    public IEnumerable<string> Roles { get; set; }

    public string Login
    {
        get { return _login; }
        set
        {
            _login = value;
            Id = 0;
            IsAuthenticated = false;
            Roles = new List<string>();
        }
    }

    public bool IsInRole(string roleName)
    {
        if (Roles == null)
            return false;

        return Roles.FirstOrDefault(r => r == roleName) != null;
    }

    public bool IsInRole(string[] roles)
    {
        return roles.Any(role => IsInRole(role));
    }
}
```

Figure 7 Transferring Data to the POCO

```
public UserEntity SelectedUser
{
    get { return _selectedUser; }
    set
    {
        _selectedUser = value;

        OnPropertyChanged("SelectedUser");
        OnPropertyChanged("FirstName");
        OnPropertyChanged("LastName");
        OnPropertyChanged("Password");
        OnPropertyChanged("PrimaryEmail");
    }
}

public string FirstName
{
    get { return _selectedUser.FirstName; }
    set
    {
        _selectedUser.FirstName = value;
        OnPropertyChanged("FirstName");
    }
}
```

UC0100-060 Events Use Case

Event aggregation, a function of Prism, provides a means to loosely couple applications while providing an excellent way to communicate between decoupled components. It does this by allowing each component to simply publish events without knowledge of the subscribers. Likewise, components can subscribe to events and handle responses without any knowledge of the publisher.

Some would argue that event aggregation makes code hard to follow, but with proper logging, it's actually the opposite. For example, I may have a drop-down list in another module that permits the user to switch the currency to use for financial values. A component I'm working on is dependent on this setting and is used to calculate the value for a ViewModel property, and there are layers of logic between my component and the component with the drop-down list (perhaps even in a separate module). If this value changes and my component doesn't get notified, it would be easier to debug using event aggregation than to trace through all of the potential logic trails to complete the path (or multiple paths) using other means. With event aggregation, there are only two points to debug: the subscriber and publisher. If I logged both (as the demo application does), it narrows a problem to a single point of failure.

The Prism event has to reside in the MsdnDemo.Phone project

Figure 8 The MessageEventArgs Handled by the MessageEvent

```
public class MessageEventArgs : EventArgs
{
    public object Sender { get; set; }
    public Enum Type { get; set; }
    public string Message { get; set; }
    public bool IsError { get; set; }
    public bool IsValid { get; set; }
    public int StatusCode { get; set; }
    public int ErrorCode { get; set; }
    private Exception _exception;
    public Exception Exception
    {
        get { return _exception; }
        set
        {
            _exception = value;
            IsError = true;
        }
    }
}
```

because of its dependencies on the Prism CompositePresentationEvent; as such, it can't reside in the PCL:

```
public class MessageEvent : CompositePresentationEvent<MessageEventArgs>
{
}
```

The EventArgs, on which the events have dependencies, are served well by the PCL because I can have a common set of event arguments—which will be used across numerous enterprise applications—within it. **Figure 8** shows the MessageEventArgs that's handled by the previous MessageEvent.

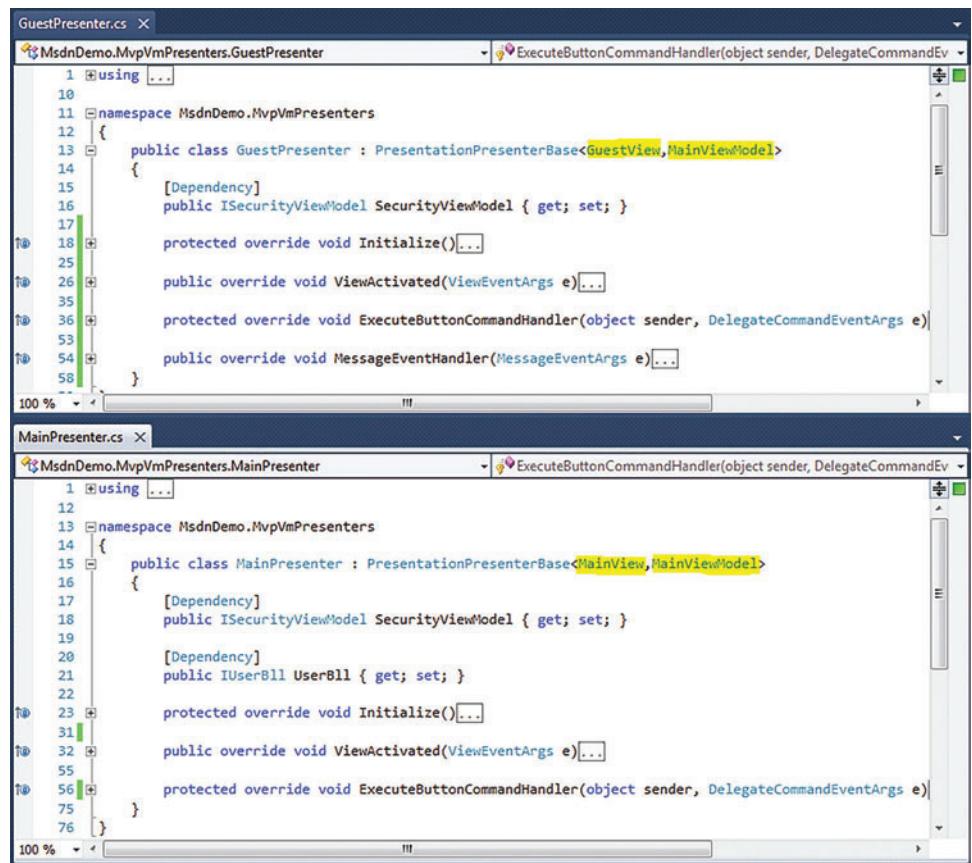


Figure 9 MsdnDemo.Phone Presenters

TRY OUR PDF SOLUTIONS FREE TODAY!

www.DynamicPDF.com/eval or call 800.681.5008 | +1 410.772.8620



WWW.DYNAMICPDF.COM

ceTe software

Figure 10 The PresentationPresenterBase

```
public class MvpVmPresenter<TView, TViewModel>
: IMvpVmPresenter
    where TView: IView
    where TViewModel : IMvpVmViewModel
{
    public IShell Shell { get; set; }
    public TView View { get; set; }
    public TViewModel ViewModel { get; set; }

    /// <summary>
    /// Called when the view is activated (by Application controller)
    /// </summary>
    public virtual void ViewActivated(ViewEventArgs e){}
}
```

UC0100-100 MvpVmBase Use Case

The highest benefit of code reuse is reliability; it's highly likely, as in the case of the MsdnDemo.Phone application, that unit tests are checking most of the code's functionality. This, combined with time in the field, will result in stable, reusable code. This will provide increased velocity and reliability for your team members, particularly when they have to reuse it for a new requirement. Additional benefits are that the code becomes easier to work on, new features can have global benefits and wiring up a new application can happen within a few hours. You simply create the Views, ViewModels and Presenters (applying applicable interfaces), and your new module can be up and running.

In the MsdnDemo.Phone application, the PresentationPresenterBase (Presenter base class) simply needs to be sent the View and ViewModel that will be used by the Presenter. Note how both Presenters share the same ViewModel in **Figure 9**.

```
17  public class PresentationPresenterBase<TView, TViewModel>
18  : MvpVmPresenter<TView, TViewModel>
19  {
20      where TView : IView
21      where TViewModel : IMvpVmViewModel
22
23      [Fields]
24
25      [Dependency]
26      public ILogger Logger{...}
27
28      [Dependency]
29      public new IShell Shell{...}
30
31      [Dependency]
32      public new TView View{...}
33
34      [Dependency]
35      public IEventAggregator EventAggregator{...}
36
37      [Dependency]
38      public new TViewModel ViewModel
39      {
40          get { return base.ViewModel; }
41          set
42          {
43              base.ViewModel = value;
44              var viewModel = value as PresentationViewModelBase;
45
46              // Wire up our button command which the ButtonStyle
47              // has a Command binding to
48              viewModel.ButtonCommand = new DelegateCommand<object>(
49                  ExecuteButtonCommandHandler,
50                  CanExecuteButtonCommandHandler);
51
52              viewModel.Presenter = this;
53
54              TryInitialize();
55          }
56      }
57  }
```

Figure 11 The Presenter Base Class

The PresentationPresenterBase derives from MvpVmPresenter, a class in my PCL, which will serve as a contract for all shared Presenters and their ViewModels across my enterprise applications. Its code is shown in **Figure 10**.

As with ILogger referenced in the PresentationModuleBase class earlier, I'll have to wrap ILogger in a similar fashion (passing the instance to the base) as well as the Shell, View and ViewModel, because these are injected via DI.

The PresentationPresenterBase, like the PresentationModuleBase, will have the responsibility of handling all DI-related services because it has references to the Prism and Unity platform-specific assemblies. As the process matures, more responsibility can be moved to their base classes in the PCL.

Note that in **Figure 11**, when the DI container resolves the specified ViewModel (TViewModel), it will set the bindings for the ButtonCommand on the ViewModel (line 99). This permits the Presenter to handle all button clicks via the ExecuteButtonCommandHandler on line 56 in **Figure 9**.

ViewModels can be shared, as is the case with MainViewModel.

Having the Presenters handle button clicks, versus the ViewModels, is one of the many benefits that inspired "olden day" architects to evolve from the Presentation-Model-Pattern and the Application-Model-Pattern to the Model-View-Presenter pattern.

ViewModels can be shared, as is the case with the MainViewModel, and each Presenter can use the MainViewModel in different ways based on its requirements. Had I moved the button clicks into the ViewModel, I would've had to use conditional statements, which in time would cause code bloat and require regression testing, as new code could affect logic for modules that weren't in the development cycle. Keeping Views and ViewModels clean of business logic allows for their maximum reuse.

Wrapping Up

The PCL can significantly reduce the number of projects required for multi-targeted applications while providing contracts for enterprise-level applications. The constraint of not being able to reference non-PCL assemblies lends itself to better coding practices, enabling projects to have a clear separation of concerns. Combining these benefits of the PCL with DI will maximize the ability to reuse decoupled projects that have been tested with time (and available unit tests), increasing the scalability, extensibility and the velocity of your team in completing new tasks.

BILL KRATOCHVIL, an independent contractor, is a lead technologist and architect for an elite team of developers working on a confidential project for a leading company in the medical industry. His own company, Global Webnet LLC, is based in Amarillo, Texas.

THANKS to the following technical experts for reviewing this article:
Christina Helton and David Kean

Does your Team do more than just track bugs?

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.

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

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

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

New in Team 2.11

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



Native
Smart Card Login
Support including
Government
and DOD

The screenshot shows a Windows Internet Explorer window titled "Service Desk Tickets - Windows Internet Explorer". The URL is "http://127.0.0.1:8080/TeamWeb/ServiceDesk.aspx". The page header includes "Alexsys Team Service Desk", "Logout", and "Welcome: John Doe". Below the header is a menu bar with "My Tickets", "New Ticket", "Edit Profile", "Knowledge Base", and "Logout". The main content area is titled "Active Tickets" and displays a table of five ticket entries:

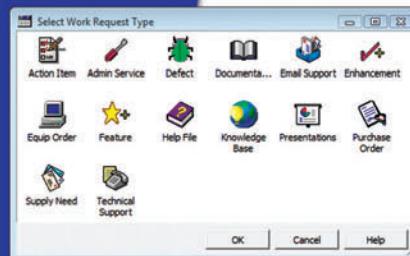
Ticket No.	Subject	Priority	Ticket Status	Open Date
SD-026554	Test of new ticket	C - Low	Open	Thu Nov 20 14:30:01 EST 2008
SD-026701	I need help setting up Team-Web	B - Moderate	Open	Fri Oct 17 09:24:32 EDT 2008
SD-026592	Custom sorts do not apply immediately in Team-Web	B - Moderate	Open	Wed Sep 24 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



Alexsys Team

Service Desk Features

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



The image displays three separate windows of the Alexsys Team application:

- New Action Item - Windows Internet Explorer**: A form for creating a new action item with fields for Title, Status (Open), Target, Next Action, and Description.
- New Technical Support - Windows Internet Explorer**: A form for creating a new technical support ticket with fields for Title, Status (Open), Customer Vendor (Alexsys Corporation), Contact (Alexsys Support), and Email.
- New Defect - Windows Internet Explorer**: A form for creating a new defect with fields for Title, Status (Open), Product (Defect), Author (ADMIN), Target, Project, Next Action, and various checkboxes for documentation, release notes, and testing.

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



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.

Particle Swarm Optimization

James McCaffrey

Particle swarm optimization (PSO) is an artificial intelligence (AI) technique that can be used to find approximate solutions to extremely difficult or impossible numeric maximization and minimization problems. The version of PSO I describe in this article was first presented in a 1995 research paper by J. Kennedy and R. Eberhart. PSO is loosely modeled on group behavior, such as bird flocking and fish schooling. The best way for you to get a feel for what PSO is and to see where I'm heading here is to examine **Figure 1**.

The first part of the figure describes a dummy problem being solved by a demonstration PSO program. The goal is to find values x_0 and x_1 so the value of the function $f = 3 + x_0^2 + x_1^2$ is minimized. Here I use the notation 2 to indicate the squaring

operation. Notice that I've deliberately chosen an unrealistically simple problem to keep the ideas of PSO clear. It's obvious that the solution to this problem is $x_0 = 0.0$ and $x_1 = 0.0$, which yields a minimum function value of 3.0, so using PSO isn't really necessary. I discuss more realistic problems that can be solved by PSO later in this article. In this example, the dimension of the function to minimize is 2 because we need to solve for 2 numeric values. In general, PSO is well-suited to numeric problems with dimensions of 2 or larger. In most situations, PSO must have some constraints on the range of possible x values. Here x_0 and x_1 are arbitrarily limited to the range -100.0 to 100.0.

In general, PSO is well-suited to numeric problems with dimensions of 2 or larger.

This article discusses:

- The AI technique of particle swarm optimization (PSO)
- Defining particles
- Understanding and implementing the PSO algorithm
- Extending a basic particle swarm optimization

Technologies discussed:

C#, Visual Studio, Microsoft .NET Framework

Code download available at:

code.msdn.microsoft.com/mag201108PSO

The next part of **Figure 1** indicates that the PSO program is using 10 particles and that the program will iterate 1,000 times. As you'll see shortly, each particle represents a possible solution to the PSO problem being solved. PSO is an iterative technique and in most cases it's not possible to know when an optimal solution has been found. Therefore, PSO algorithms usually must have some limit on the number of iterations to perform.

```

Visual Studio 2008 Command Prompt
C:\ParticleSwarmOptimization\bin\Debug>ParticleSwarmOptimization.exe
Begin Particle Swarm Optimization demonstration

Objective function to minimize has dimension = 2
Objective function is f(x) = 3 + <x0^2 + x1^2>
Range for all x values is -100 <= x <= 100

Number iterations = 1000
Number particles in swarm = 10

Initializing swarm with random positions/solutions
Initialization complete
Initial best fitness = 744.1182
Best initial position/solution:
x0 = 26.5318
x1 = -6.0976

Entering main PSO processing loop
Processing complete
Final best fitness = 3.0000
Best position/solution:
x0 = 0.0000
x1 = 0.0000

End PSO demonstration

C:\ParticleSwarmOptimization\bin\Debug>

```

Figure 1 Particle Swarm Optimization Demo Run

The next lines in **Figure 1** indicate that each of the 10 particles in the swarm is initialized to a random position. A particle's position represents a possible solution to the optimization problem to be solved. The best randomly generated initial position is $x_0 = 26.53$ and $x_1 = -6.09$, which corresponds to a fitness (the measure of solution quality) of $3 + 26.53^2 + (-6.09)^2 = 744.12$. The PSO algorithm then enters a main processing loop where each particle's position is updated on each pass through the loop. The update procedure is the heart of PSO and I'll explain it in detail later in this article. After 1,000 iterations, the PSO algorithm did in fact find the optimal solution of $x_0 = 0.0$ and $x_1 = 0.0$, but let me emphasize that in most situations you won't know whether a PSO program has found an optimal solution.

In this article, I'll explain in detail the PSO algorithm and walk you line by line through the program shown running in **Figure 1**. I coded the demo program in C#, but you should be able to easily adapt the code presented here to another language, such as Visual Basic .NET or Python. The complete source code for the program presented in this article is available at code.msdn.microsoft.com/mag201108PSO. This article assumes you have intermediate coding skills with a modern procedural language but does not assume you know anything about PSO or related AI techniques.

Particles

When using PSO, a possible solution to the numeric optimization problem under investigation is represented by the position of a particle. Additionally, each particle has a current velocity, which represents a magnitude and direction toward a new, presumably better, solution/position. A particle also has a measure of the quality of its current position, the particle's best known position (that is, a previous position with the best known quality), and the quality of the best known position. I coded a Particle class as shown in **Figure 2**.

The Particle class has five public data members: position, fitness, velocity, bestPosition and bestFitness. When using PSO, for simplicity

I prefer using public scope fields, but you may want to use private fields along with get and set properties instead. The field named position is an array of type double and represents a possible solution to the optimization problem under investigation. Although PSO can be used to solve non-numeric problems, it's generally best-suited for solving numeric problems. Field fitness is a measure of how good the solution represented by position is. For minimization problems, which are the most common types of problems solved by PSO, smaller values of the fitness field are better than larger values; for maximization problems, larger values of fitness are better.

Field velocity is an array of type double and represents the information necessary to update a particle's current position/solution.

I'll explain particle velocity in detail shortly. The fourth and fifth fields in the Particle type are bestPosition and bestFitness. These fields hold the best position/solution found by the Particle object and the associated fitness of the best position.

The Particle class has a single constructor that accepts five parameters that correspond to each of the Particle's five data fields. The constructor simply copies each parameter value to its corresponding data field. Because all five Particle fields have public scope, I could have omitted the constructor and then just used field assignment statements in the PSO code, but I think the constructor leads to cleaner code.

I coded the demo program in C#, but you should be able to easily adapt the code to another language.

The Particle class definition contains a ToString method that echoes the values of the five data fields. As with the constructor, because I declared the position, fitness, velocity, bestPosition and bestFitness fields with public scope, I don't really need a ToString method to view a Particle object's values, but including it simplifies viewing the fields and it's useful for WriteLine-style debugging during development. In the ToString method I use string concatenation rather than the more efficient StringBuilder class to make it easier for you to refactor my code to a non-Microsoft .NET Framework-based language if you wish.

The PSO Algorithm

Although the heart of the PSO algorithm is rather simple, you'll need to understand it thoroughly in order to modify the code in this article to meet your own needs. PSO is an iterative process. On each iteration in the PSO main processing loop, each particle's current velocity is first updated based on the particle's current velocity, the particle's local information and global swarm information. Then, each particle's position is updated using the particle's new velocity. In math terms the two update equations are:

$$\begin{aligned} v(t+1) &= (w * v(t)) + (c1 * r1 * (p(t) - x(t)) + (c2 * r2 * (g(t) - x(t))) \\ x(t+1) &= x(t) + v(t+1) \end{aligned}$$

Bear with me here; the position update process is actually much simpler than these equations suggest. The first equation updates a particle's velocity. The term $v(t+1)$ means the velocity at time $t+1$. Notice that v is in bold, indicating that velocity is a vector value and has multiple components such as $\{1.55, -0.33\}$, rather than being a single scalar value. The new velocity depends on three terms. The first term is $w * v(t)$. The w factor is called the inertia weight and is simply a constant like 0.73 (more on this shortly); $v(t)$ is the current velocity at time t . The second term is $c1 * r1 * (p(t) - x(t))$. The $c1$ factor is a constant called the cognitive (or personal or local) weight. The $r1$ factor is a random variable in the range $[0, 1]$, which is greater than or equal to 0 and strictly less than 1.

Figure 2 Particle Definition

```
public class Particle
{
    public double[] position;
    public double fitness;
    public double[] velocity;

    public double[] bestPosition;
    public double bestFitness;

    public Particle(double[] position, double fitness,
        double[] velocity, double[] bestPosition, double bestFitness)
    {
        this.position = new double[position.Length];
        position.CopyTo(this.position, 0);
        this.fitness = fitness;
        this.velocity = new double[velocity.Length];
        velocity.CopyTo(this.velocity, 0);
        this.bestPosition = new double[bestPosition.Length];
        bestPosition.CopyTo(this.bestPosition, 0);
        this.bestFitness = bestFitness;
    }

    public override string ToString()
    {
        string s = "";
        s += "=====\n";
        s += "Position: ";
        for (int i = 0; i < this.position.Length; ++i)
            s += this.position[i].ToString("F2") + " ";
        s += "\n";
        s += "Fitness = " + this.fitness.ToString("F4") + "\n";
        s += "Velocity: ";
        for (int i = 0; i < this.velocity.Length; ++i)
            s += this.velocity[i].ToString("F2") + " ";
        s += "\n";
        s += "Best Position: ";
        for (int i = 0; i < this.bestPosition.Length; ++i)
            s += this.bestPosition[i].ToString("F2") + " ";
        s += "\n";
        s += "Best Fitness = " + this.bestFitness.ToString("F4") + "\n";
        s += "=====\n";
        return s;
    }
} // class Particle
```

The $p(t)$ vector value is the particle's best position found so far. The $x(t)$ vector value is the particle's current position. The third term in the velocity update equation is $(c2 * r2 * (g(t) - x(t)))$. The $c2$ factor is a constant called the social—or global—weight. The $r2$ factor is a random variable in the range $[0, 1]$. The $g(t)$ vector value is the best known position found by any particle in the swarm so far. Once the new velocity, $v(t+1)$, has been determined, it's used to compute the new particle position $x(t+1)$.

When using PSO, for simplicity I prefer using public scope fields.

A concrete example will help make the update process clear. Suppose you're trying to minimize $3 + x_0^2 + x_1^2$ as described in the introductory section of this article. The function is plotted in **Figure 3**. The base of the containing cube in **Figure 3** represents x_0 and x_1 values and the vertical axis represents the function value. Note that the plot surface is minimized with $f = 3$ when $x_0 = 0$ and $x_1 = 0$.

Let's say that a particle's current position, $x(t)$, is $\{x_0, x_1\} = \{3.0, 4.0\}$, and that the particle's current velocity, $v(t)$, is $\{-1.0, -1.5\}$. Let's also assume that constant $w = 0.7$, constant $c1 = 1.4$, constant $c2 = 1.4$, and that random numbers $r1$ and $r2$ are 0.5 and 0.6 respectively. Finally, suppose the particle's best known position is $p(t) = \{2.5, 3.6\}$ and the global best known position by any particle in the swarm is $g(t) = \{2.3, 3.4\}$. Then the new velocity and position values are:

$$\begin{aligned} v(t+1) &= (0.7 * \{-1.0, -1.5\}) + \\ &\quad (1.4 * 0.5 * \{2.5, 3.6\} - \{3.0, 4.0\}) + \\ &\quad (1.4 * 0.6 * \{2.3, 3.4\} - \{3.0, 4.0\}) \\ &= \{-0.70, -1.05\} + \{-0.35, -0.28\} + \{-0.59, -0.50\} \\ &= \{-1.64, -1.83\} \\ x(t+1) &= \{3.0, 4.0\} + \{-1.64, -1.83\} \\ &= \{1.36, 2.17\} \end{aligned}$$

Recall that the optimal solution is $\{x_0, x_1\} = \{0.0, 0.0\}$. Observe that the update process has improved the old position/solution from $\{3.0, 4.0\}$ to $\{1.36, 2.17\}$. If you mull over the update process a bit, you'll see that the new velocity is the old velocity (times a weight) plus a factor that depends on a particle's best known position, plus another factor that depends on the best known position from all particles in the swarm. Therefore, a particle's new position tends to move toward a better position based on the particle's best known position and the best known position of all particles. The graph in **Figure 4** shows the movement of one of the particles during the first eight iterations of the demo PSO run. The particle starts at $x_0 = 100.0$, $x_1 = 80.4$ and tends to move toward the optimal solution of $x_0 = 0$, $x_1 = 0$. The spiral motion is typical of PSO.

Implementing the PSO Algorithm

Figure 5 presents the overall structure of the PSO program that produced the program run shown in **Figure 1**.

I used Visual Studio to create a C# console application project named ParticleSwarmOptimization. PSO code is fairly basic, so any version of the .NET Framework (1.1 through 4) will work well. I removed all Visual Studio-generated using statements except for the reference to the core System namespace. I declared a class-scope

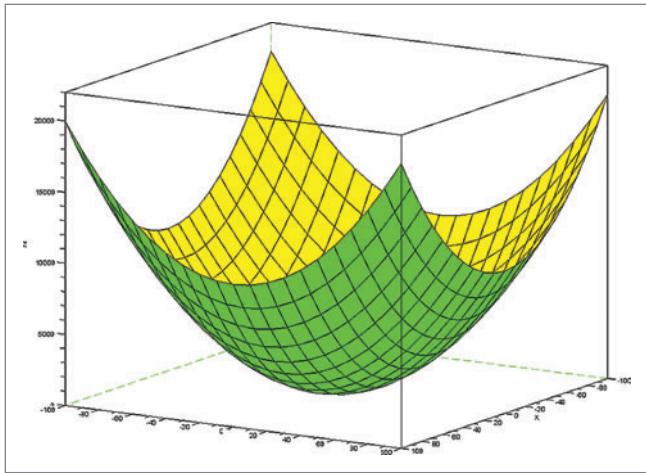


Figure 3 Plot of $f = 3 + x_0^2 + x_1^2$

object of type Random to generate the cognitive and social random numbers described in the previous section. I also used the Random object to generate random initial velocities and positions for each Particle object. Inside the Main method I wrap all my code in a single, high-level try statement to catch any exceptions.

After instantiating the Random object with an arbitrary seed value of 0, I initialize some key PSO variables:

```
int numberParticles = 10;
int numberIterations = 1000;
int iteration = 0;
int Dim = 2;
double minX = -100.0;
double maxX = 100.0;
```

I use 10 Particle objects. As a rule of thumb, more Particle objects are better than fewer, but more can significantly slow program performance. I set the number of main processing loop iterations to 1,000. The number of iterations you'll want to use will depend on the complexity of the problem you're trying to optimize and the processing power of your host machine. Typically, PSO programs use a value between 1,000 and 100,000. The variable named iteration is a counter to keep track of the number of main loop iterations. The Dim variable holds the number of x values in a solution/position. Because my example problem needs to find

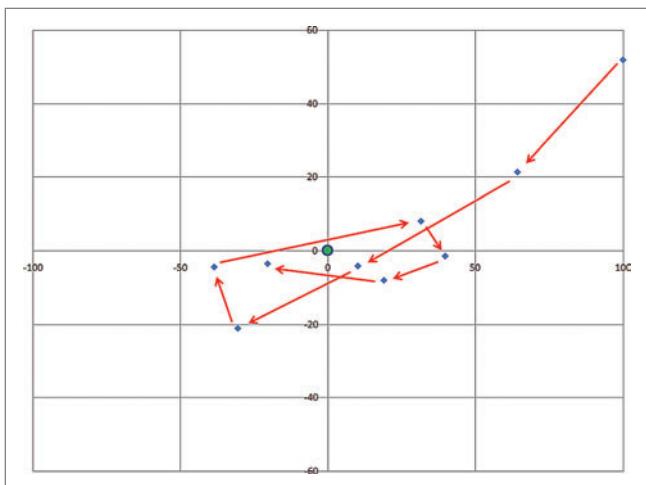


Figure 4 Particle Motion Toward Optimal Solution

the values of x_0 and x_1 that minimize $3 + x_0^2 + x_1^2$, I set Dim to 2. As I mentioned earlier, in most PSO situations you'll want to limit the x values that make up the position/solution vector to some problem-dependent range. Without some limits, you're effectively searching from double.MinValue to double.MaxValue. Here I arbitrarily limit x_0 and x_1 to $[-100.0, +100.0]$.

Next, I prepare to instantiate the particle swarm:

```
Particle[] swarm = new Particle[numberParticles];
double[] bestGlobalPosition = new double[Dim];
double bestGlobalFitness = double.MaxValue;
double minV = -1.0 * maxX;
double maxV = maxX;
```

I create an array of Particle objects named swarm. I also set up an array to hold the global best known position determined by any Particle—denoted by $g(t)$ in the algorithm—and the corresponding fitness of that position array. I set constraints for the maximum and minimum values for a new velocity. The idea here is that because a new velocity determines a particle's new position, I don't want the magnitude of any of the velocity components to be huge.

The code to initialize the swarm is as follows:

```
for (int i = 0; i < swarm.Length; ++i)
{
    double[] randomPosition = new double[Dim];
    for (int j = 0; j < randomPosition.Length; ++j) {
        double lo = minX;
        double hi = maxX;
        randomPosition[j] = (hi - lo) * ran.NextDouble() + lo;
    }
    ...
}
```

I iterate through each Particle object in the array named swarm. I declare an array of size Dim to hold a random position for the current Particle. Then for each x-value of the position I generate a random value between minX (-100.0) and maxX (+100.0). In many realistic PSO problems, the range for each x-value will be different, so you'll have to add code to deal with each x-value in the position array separately.

Now I continue the initialization process:

```
double fitness = ObjectiveFunction(randomPosition);
double[] randomVelocity = new double[Dim];
for (int j = 0; j < randomVelocity.Length; ++j) {
    double lo = -1.0 * Math.Abs(maxX - minX);
    double hi = Math.Abs(maxX - minX);
    randomVelocity[j] = (hi - lo) * ran.NextDouble() + lo;
}
swarm[i] = new Particle(randomPosition, fitness, randomVelocity,
    randomPosition, fitness);
...

```

First I compute the quality of the current random position array by passing that array to the method ObjectiveFunction. If you refer back to **Figure 5**, you'll see that the ObjectiveFunction method simply computes the value of the function I'm trying to minimize, namely $3 + x_0^2 + x_1^2$. Next I compute a random velocity for the current Particle object. After I have a random position, the fitness of the random position and a random velocity, I pass those values to the Particle constructor. Recall that the fourth and fifth parameters are the particle's best known position and its associated fitness, so when initializing a Particle the initial random position and fitness are the best known values.

The swarm initialization code finishes with:

```
...
if (swarm[i].fitness < bestGlobalFitness) {
    bestGlobalFitness = swarm[i].fitness;
    swarm[i].position.CopyTo(bestGlobalPosition, 0);
}
} // End initialization loop
```

I check to see if the fitness of the current Particle is the best (smallest in the case of a minimization problem) fitness found so far. If so, I update array bestGlobalPosition and the corresponding variable bestGlobalFitness.

Next, I prepare to enter the main PSO processing loop:

```
double w = 0.729; // inertia weight
double c1 = 1.49445; // cognitive weight
double c2 = 1.49445; // social weight
double r1, r2; // randomizers
```

I set the value for w, the inertia weight, to 0.729. This value was recommended by a research paper that investigated the effects of various PSO parameter values on a set of benchmark minimization problems. Instead of a single, constant value for w, an alternative approach is to vary the value of w. For example, if your PSO algorithm is set to iterate 10,000 times, you could initially set w to 0.90 and gradually decrease w to 0.40 by reducing w by 0.10 after

Figure 5 PSO Program Structure

```
using System;
namespace ParticleSwarmOptimization
{
    class Program
    {
        static Random ran = null;
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("\nBegin PSO demo\n");
                ran = new Random(0);

                int numberParticles = 10;
                int numberIterations = 1000;
                int iteration = 0;
                int Dim = 2; // dimensions
                double minX = -100.0;
                double maxX = 100.0;

                Particle[] swarm = new Particle[numberParticles];
                double[] bestGlobalPosition = new double[Dim];
                double bestGlobalFitness = double.MaxValue;

                double minV = -1.0 * maxX;
                double maxV = maxX;

                // Initialize all Particle objects

                double w = 0.729; // inertia weight
                double c1 = 1.49445; // cognitive weight
                double c2 = 1.49445; // social weight
                double r1, r2; // randomizations

                // Main processing loop

                // Display results
                Console.WriteLine("\nEnd PSO demo\n");
            }
            catch (Exception ex)
            {
                Console.WriteLine("Fatal error: " + ex.Message);
            }
        } // Main()

        static double ObjectiveFunction(double[] x)
        {
            return 3.0 + (x[0] * x[0]) + (x[1] * x[1]);
        }

    } // class Program

    public class Particle
    {
        // Definition here
    }

} // ns
```

every 2,000 iterations. The idea of a dynamic w is that early in the algorithm you want to explore larger changes in position, but later on you want smaller particle movements. I set the values for both c1 and c2, the cognitive and social weights, to 1.49445. Again, this value was recommended by a research study. If you set the value of c1 to be larger than the value of c2, you place more weight on a particle's best known position than on the swarm's global best known position, and vice versa. The random variables r1 and r2 add a random component to the PSO algorithm and help prevent the algorithm from getting stuck at a non-optimal local minimum or maximum solution.

Next, I begin the main PSO processing loop:

```
for (int i = 0; i < swarm.Length; ++i)
{
    Particle currP = swarm[i];

    for (int j = 0; j < currP.velocity.Length; ++j)
    {
        r1 = ran.NextDouble();
        r2 = ran.NextDouble();

        newVelocity[j] = (w * currP.velocity[j]) +
            (c1 * r1 * (currP.bestPosition[j] - currP.position[j])) +
            (c2 * r2 * (bestGlobalPosition[j] - currP.position[j]));
    }
}
```

I iterate through each Particle object in the swarm array using i as an index variable. I create a reference to the current Particle object named currP to simplify my code, but I could have used swarm[i] directly. As explained in the previous section, the first step is to update each particle's velocity vector. For the current Particle object, I walk through each one of the values in the object's velocity array, generate random variables r1 and r2, and then update each velocity component as explained in the previous section.

After I compute a new velocity component for the current Particle object, I check to see if that component is between the minimum and maximum values for a velocity component:

```
if (newVelocity[j] < minV)
    newVelocity[j] = minV;
else if (newVelocity[j] > maxV)
    newVelocity[j] = maxV;
} // each j
newVelocity.CopyTo(currP.velocity, 0);
...
```

If the component is out of range, I bring it back in range. The idea here is that I don't want extreme values for the velocity component because extreme values could cause my new position to spin out of bounds. After all velocity components have been computed, I update the current Particle object's velocity array using the handy .NET CopyTo method.

Once the velocity of the current Particle has been determined, I can use the new velocity to compute and update the current Particle's position:

```
for (int j = 0; j < currP.position.Length; ++j)
{
    newPosition[j] = currP.position[j] + newVelocity[j];
    if (newPosition[j] < minX)
        newPosition[j] = minX;
    else if (newPosition[j] > maxX)
        newPosition[j] = maxX;
}
newPosition.CopyTo(currP.position, 0);
...
```

Again I perform a range check, this time on each of the current particle's new position components. In a sense, this is a redundant check because I've already constrained the value of each velocity component, but in my opinion the extra check is warranted here.

Now that I have the current Particle object's new position, I compute the new fitness value and update the object's fitness field:

```
newFitness = ObjectiveFunction(newPosition);
currP.fitness = newFitness;

if (newFitness < currP.bestFitness) {
    newPosition.CopyTo(currP.bestPosition, 0);
    currP.bestFitness = newFitness;
}
if (newFitness < bestGlobalFitness) {
    newPosition.CopyTo(bestGlobalPosition, 0);
    bestGlobalFitness = newFitness;
}

} // each Particle
} // main PSO loop
...
```

After updating the current particle, I check to see if the new position is the best known position of the particle; I also check to see if the new position is a best global swarm position. Notice that logically, there can be a new global best position only if there's a best local position, so I could have nested the global best check inside the check for a local best position.

At this point my main PSO algorithm loop is finished and I can display my results:

```
Console.WriteLine("\nProcessing complete");
Console.WriteLine("Final best fitness = ");
Console.WriteLine(bestGlobalFitness.ToString("F4"));
Console.WriteLine("Best position/solution:");
for (int i = 0; i < bestGlobalPosition.Length; ++i){
    Console.Write("x" + i + " = ");
    Console.WriteLine(bestGlobalPosition[i].ToString("F4") + " ");
}
Console.WriteLine("");
Console.WriteLine("\nEnd PSO demonstration\n");
}
catch (Exception ex)
{
    Console.WriteLine("Fatal error: " + ex.Message);
}
} // Main()
```

Extending and Modifying

Now that you've seen how to write a basic PSO, let's discuss how you can extend and modify the code I've presented. The example problem I solved is artificial in the sense that there's no need to use PSO to find an approximate solution because the problem can be solved exactly. Where PSO is really useful is when the numeric problem under investigation is extremely difficult or impossible to solve using standard techniques. Consider the following problem. You want to predict the score of an (American) football game between teams A and B. You have historical data consisting of the previous results of A and B against other teams. You mathematically model the historical rating of a team X in such a way that if the team wins a game, the team's rating goes up by some fixed value (say 16 points) plus another value that depends on the difference between the teams' ratings (say 0.04 times the difference if the team X rating is less than the opposing team's). Furthermore, you model the predicted margin of victory of a team as some function of the difference in team ratings; for example, if team X is rated 1,720 and team Y is rated 1,620, your model predicts a margin of victory for X of 3.5 points. In short, you have a large amount of data and need to determine several numeric values (such as the 16 and the 0.04) that minimize your prediction errors. This data-driven parameter estimation is the type of problem that's right up PSO's alley.

PSO is just one of several AI techniques based on the behavior of natural systems. Perhaps the technique closest to PSO algorithms is Genetic Algorithms (GAs). Both techniques are well-suited to difficult numeric problems. GAs have been extensively studied for decades. An advantage of PSOs over GAs is that PSO algorithms are significantly simpler to implement than GAs. It's not clear at this time whether PSOs are more or less effective than GAs, or roughly equal to them.

The version of PSO I've presented here can be modified in many ways. One particularly interesting modification is to use several sub-swarms of particles rather than one global swarm. With such a design, each particle belongs to a sub-swarm and the new velocity of a particle could depend on four terms rather than three: the old velocity, the particle's best known position, the best known position of any particle in the sub-swarm, and the best known position of any particle. The idea of this sub-swarm design is to reduce the chances of the PSO algorithm getting stuck in a non-optimal solution. To the best of my knowledge such a design has not yet been thoroughly investigated. ■

DR. JAMES MCCAFFREY works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He's worked on several Microsoft products, including Internet Explorer and MSN Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006), and can be reached at jammcc@microsoft.com.

THANKS to the following Microsoft technical experts for reviewing this article: Paul Koch, Dan Liebling, Anne Loomis Thompson and Shane Williams

Add powerful diagramming capabilities to your applications in less time than you ever imagined with GoDiagram components.

GoDiagram

The first and still the best. We were the first to create diagram controls for .NET and we continue to lead the industry.

Fully customizable interactive diagram components save countless hours of programming enabling you to build applications in a fraction of the time.

Our new WPF and Silverlight products fully support XAML, including data-binding, templates, and styling.

For .NET WinForms, ASP.NET, WPF and Silverlight
Specializing in diagramming products for programmers for 15 years!

Powerful, flexible, and easy to use.
Find out for yourself with our **FREE** Trial Download with full support at: www.godialogram.com



Font Metrics in Silverlight

Most graphical programming environments have classes or functions to obtain font metrics. These font metrics provide information about the size of text characters when rendered with a particular font. At the very least, the font metrics information includes the widths of all the individual characters and a height that's common to all characters. Internally, these widths are probably stored in an array, so the access is very fast. Font metrics are invaluable for laying out text in paragraphs and pages.

Unfortunately, Silverlight is one graphical environment that does *not* provide font metrics to application program developers. If you wish to obtain the size of text prior to rendering it, you must use `TextBlock`, which is, of course, the same element you use for rendering text. Internally, `TextBlock` obviously has access to font metrics; otherwise it would have no idea how large the text is supposed to be.

Font metrics are invaluable
for laying out text in paragraphs
and pages.

It's easy to persuade `TextBlock` to provide text dimensions without actually rendering the text. Simply instantiate a `TextBlock` element, initialize the `FontFamily`, `FontSize`, `FontStyle`, `FontWeight` and `Text` properties, and then query the `ActualWidth` and `ActualHeight` properties. Unlike some Silverlight elements, you don't need to make this `TextBlock` a child of a `Panel` or `Border`. Nor do you need to call the `Measure` method on the parent.

To speed up this process, you can use `TextBlock` to build an array of character widths, and then you can use this array to mimic traditional font metrics. This is what I'll show you how to do in this article.

Is All This Really Necessary?

Most Silverlight programmers don't mourn the absence of font metrics because, for many applications, `TextBlock` makes them unnecessary. `TextBlock` is very versatile. If you use the `Inlines` property, a single `TextBlock` element can render a mix of italic and bold

text, and even text with different font families or font sizes. `TextBlock` can also wrap long text into multiple lines to create paragraphs. It's this text-wrapping feature that I've been using in the past two installments of this column to create simple e-book readers for Windows Phone 7.

In the previous issue, I presented a program called `Middle-marchReader` that lets you read George Eliot's novel "Middlemarch" on your phone. I want you to perform an experiment with that program: Deploy a fresh copy on an actual Windows Phone 7 device. (If necessary, first uninstall any version that might be on the phone already.) Now press the application bar button to get the list of chapters. Choose chapter IV. From the first page of chapter IV, flick your finger on the screen from left to right to go to the last page of the third chapter and start counting seconds: "One Mississippi, two Mississippi..."

If your phone is anything like my phone, you'll discover that paging back from the beginning of Chapter IV to the end of Chapter III takes about 10 seconds. I think we can all agree that 10 seconds is much too long for a simple page turn!

This wait is characteristic of on-the-fly pagination. Displaying the last page of a chapter requires that all the previous pages in that chapter be paginated first. I've mentioned in previous installments of this column that the pagination technique I've been using is grossly inefficient, and this particular example proves it.

My slow pagination technique uses the text-wrapping feature of `TextBlock` to render entire paragraphs, or partial paragraphs if the paragraph straddles multiple pages. If a paragraph is too large to fit on a page, then my code starts lopping off words at the end of the paragraph until it fits. After each word is removed, Silverlight must re-measure the `TextBlock`, and this requires lots of time.

Certainly I need to revise my pagination logic. A better pagination algorithm breaks each paragraph into individual words, obtains the size of each word, and performs its own word-positioning and line-wrapping logic.

In the previous e-book readers I've shown in this column, each paragraph (or partial paragraph) on a page is just one `TextBlock`, and these `TextBlock` elements are children of a `StackPanel`. In the e-book reader I'll describe in this column, every word on the page is its own `TextBlock`, and each `TextBlock` is positioned at a specific location on a `Canvas`. These multiple `TextBlock` elements require a little more time for Silverlight to render the page, but the page layout is speeded up enormously. My experiments show that the troublesome 10-second page transition in `MiddlemarchReader`

Code download available at code.msdn.microsoft.com/mag201108UIFrontiers.

Figure 1 The FontMetrics Class

```
public class FontMetrics
{
    const int EmSize = 2048;
    TextBlock txtblk;
    double height;
    double[][] charWidths = new double[256][];
}

public FontMetrics(Font font)
{
    this.Font = font;

    // Create the TextBlock for all measurements
    txtblk = new TextBlock
    {
        FontFamily = this.Font.FontFamily,
        FontStyle = this.Font.FontStyle,
        FontWeight = this.Font.FontWeight,
        FontSize = EmSize
    };

    // Store the character height
    txtblk.Text = " ";
    height = txtblk.ActualHeight / EmSize;
}

public Font Font { protected set; get; }

public double this[char ch]
{
    get
    {
        // Break apart the character code
        int upper = (ushort)ch >> 8;
        int lower = (ushort)ch & 0xFF;

        // If there's no array, create one
        if (charWidths[upper] == null)
    }
}

{
    charWidths[upper] = new double[256];

    for (int i = 0; i < 256; i++)
        charWidths[upper][i] = -1;
}

// If there's no character width, obtain it
if (charWidths[upper][lower] == -1)
{
    txtblk.Text = ch.ToString();
    charWidths[upper][lower] = txtblk.ActualWidth / EmSize;
}
return charWidths[upper][lower];
}

public Size MeasureText(string text)
{
    double accumWidth = 0;

    foreach (char ch in text)
        accumWidth += this[ch];

    return new Size(accumWidth, height);
}

public Size MeasureText(string text, int startIndex, int length)
{
    double accumWidth = 0;

    for (int index = startIndex; index < startIndex + length; index++)
        accumWidth += this[text[index]];

    return new Size(accumWidth, height);
}
```

is reduced to two seconds when each word is measured with a *TextBlock* element, and to 0.5 seconds when character widths are cached in an array like traditional font metrics.

But it's time for a new book. The downloadable Visual Studio solution for this article is called PhineasReader and it lets you read the story of one of Anthony Trollope's most beloved fictional characters, the Irish Member of Parliament, "Phineas Finn" (1869). Once again, I've used a plain-text file downloaded from Project Gutenberg (gutenberg.org).

Silverlight is one graphical environment that does not provide font metrics to application program developers.

The FontMetrics Class

When a computer font is first designed, the font designer chooses a number that's called the "em-size." The term comes from olden days when the capital letter M was a square block of type, and the size of that M determined the heights and relative widths of all the other characters.

Many TrueType fonts are designed with an em-size of 2,048 "design units." That size is large enough so that the character height is

an integer—usually greater than 2,048 to accommodate diacritic marks—and all the widths of all the characters are integers as well.

If you create a *TextBlock* using any of the fonts supported on Windows Phone 7, and set the *FontSize* property to 2,048, you'll discover that *ActualWidth* returns an integer regardless what character you set to the *Text* property. (*ActualHeight* is also an integer except for the Segoe WP Bold font and the default Portable User Interface font. These two names refer to the same font, and the height is 2,457.6. I don't know the reason for this inconsistency.)

Once you obtain the character height and widths based on a *FontSize* property set to 2,048, you can simply scale that height and the widths for any other font size.

Figure 1 shows the *FontMetrics* class I created. If you need to deal with multiple fonts, you'd maintain a separate *FontMetrics* instance for each font family, font style (regular or italic) and font weight (regular or bold). It's quite likely these *FontMetrics* instances would be referenced from a dictionary, so I created a *Font* class that implements the *IEquatable* interface, hence it's suitable as a dictionary key. My e-book reader only needs one *FontMetrics* instance based on the default Windows Phone 7 font.

Originally I thought I would take advantage of my knowledge about the common em-size of 2,048 and store all character widths as integers, perhaps 16-bit integers. However, I decided to play it safe and store them as double-precision floating-point values instead. I then decided that *FontMetrics* would divide the *ActualWidth* and *ActualHeight* values by 2,048, so it really stores values appropriate for a *FontSize* of 1. This makes it easy for any program using the class to multiply the values by the desired *FontSize*.

The Project Gutenberg plain-text files only contain characters with Unicode values less than 256. Therefore, the FontMetrics class could store all the character widths it needs in a simple array of 256 values. Because this class might be used for text with character codes greater than 255, I wanted something more flexible than that, but I knew that the last thing I wanted was to allocate an array sufficient to store 64,536 double-precision floating point values. That's .5MB of memory just for the font metrics!

Instead, I used a jagged array. The array named charWidths has 256 elements, each of which is an array of 256 double values. A 16-bit character code is divided into two 8-bit indices. The upper byte indexes the charWidths array to obtain an array of 256 double values, and then the lower byte of the character code indexes that array. But these arrays of double values are only created as they're needed, and individual character widths are obtained only as they're needed. This logic takes place in the indexer of the FontMetrics class, and both reduces the amount of storage required by the class and cuts down unnecessary processing for characters that are never used.

The two MeasureText methods obtain the size of a string, or a substring of a larger string. These two methods return values appropriate for a FontSize of 1, which can then be scaled simply by multiplying by the desired font size.

TextBlock elements are usually aligned on pixel boundaries because the UseLayoutRounding property defined by the UIElement class has a default value of *true*. For text, pixel alignment helps readability because it avoids inconsistent anti-aliasing. After multiplying the values obtained from MeasureText by the font size, you'll want to pass those values through the Math.Ceiling method. This will give you values rounded up to the next integral pixel.

Fancier Formatting

As in my previous e-book readers, most of the real grunt work of the program occurs in the PageProvider class. This class has two main jobs: pre-processing the Project Gutenberg file to concatenate individual lines of the file into single-line paragraphs, and pagination.

To test FontMetrics for character codes greater than 255, I decided to perform a little bit more pre-processing than in the past. First, I replaced standard double quotes (ASCII code 0x22) with “fancy quotes” (Unicode 0x201C and 0x201D) by simply alternating the two codes within each paragraph. Also, Victorian authors tend to use a lot of em-dashes—often to delimit phrases like this one—and these turn up in the Project Gutenberg files as pairs of dashes. In most cases, I replaced these pairs of dashes with

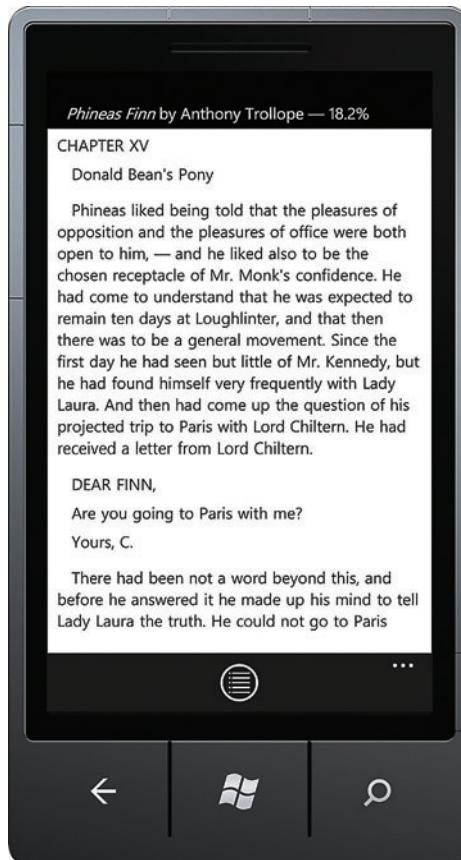


Figure 2 A Page from PhineasReader Showing Paragraph Indenting

Unicode 0x2014 surrounded by spaces to facilitate line breaks.

My new pre-processing logic also handles consecutive lines with the same indenting. Often these indented lines comprise a letter or other indented material in the text, and I tried to handle those in a more graceful way. While paginating, I began all non-indented paragraphs with a first-line indent, except for the first paragraph of a chapter, which I presume is usually a chapter title.

The overall effect of this indentation logic is illustrated in **Figure 2**.

Pagination and Composition

Because PageProvider has taken over much of the layout previously performed by TextBlock itself, the pagination logic has become a little too lengthy for the pages of this magazine. But it's fairly straightforward. All the paragraphs that comprise the Project Gutenberg text are stored as a List of ParagraphInfo objects. The formatted book is a BookInfo object that's mostly a List of ChapterInfo objects. The ChapterInfo object indicates the index of the paragraph that begins the chapter and also maintains a List of PageInfo objects that are created as the book is progressively paginated.

The PageInfo class is shown in **Figure 3**. It indicates where the page begins with a paragraph index and a character index within that paragraph, and also maintains a List of WordInfo objects. The WordInfo class is shown in **Figure 4**. Each WordInfo object corresponds to a single word, so this class indicates the word's coordinate location on the page and the text of the word as a substring of a paragraph.

Most Silverlight programmers don't mourn the absence of font metrics because, for many applications, TextBlock makes them unnecessary.

You'll notice in the PageInfo class that the Words property is flagged with XmlIgnore, meaning that this property won't be serialized with the rest of the class, and hence isn't saved in isolated storage along with the rest of the pagination information. A few little calculations will convince you of the wisdom of this decision:

Figure 3 The PageInfo Class Represents Each Paginated Page

```
public class PageInfo
{
    public PageInfo()
    {
        this.Words = new List<WordInfo>();
    }

    public int ParagraphIndex { set; get; }

    public int CharacterIndex { set; get; }

    public bool IsLastPageInChapter { set; get; }

    public bool IsPaginated { set; get; }

    public int AccumulatedCharacterCount { set; get; }

    [XmlAttribute]
    public List<WordInfo> Words { set; get; }
}
```

"Phineas Finn" is more than 200,000 words in length, and WordInfo contains 20 bytes of data, so, in memory, all the WordInfo objects will occupy more than 4MB. That's not too bad, but consider these 200,000 WordInfo objects converted to XML for serialization! Besides, if the beginning of a page is known, calculating the locations of the words on that page using the FontMetrics class is very fast, so these WordInfo objects can be recreated without performance problems.

Figure 5 shows the BuildPageElement method in PageProvider that basically converts a PageInfo object into a Canvas containing a bunch of TextBlock elements. It's this Canvas that's actually rendered on the screen.

The actual pagination and layout code doesn't touch the UI. Only the BuildPageElement method that composes the page creates UI objects. The separation of pagination from page composition is new in this version of the e-book reader, and it means that the pagination and layout could occur in a background thread. I'm not doing that in this program, but it's something to keep in mind.

Not Just for Performance

I originally decided to abandon TextBlock for layout for performance reasons. But there are at least two more compelling reasons for using separate TextBlock elements for each word.

First, if you ever wanted to justify your paragraphs, this is an essential first step. Silverlight for Windows Phone 7 doesn't support the TextAlign.Justify enumeration member. But if every word is a separate TextBlock, justification is simply a matter of distributing extra space between the individual words.

Figure 4 The WordInfo Class Represents a Single Word

```
public class WordInfo
{
    public int LocationLeft { set; get; }

    public int LocationTop { set; get; }

    public int ParagraphIndex { set; get; }

    public int CharacterIndex { set; get; }

    public int CharacterCount { set; get; }
}
```

Figure 5 The BuildPageElement Method in PageProvider

```
FrameworkElement BuildPageElement(ChapterInfo chapter, PageInfo pageInfo)
{
    if (pageInfo.Words.Count == 0)
    {
        Paginate(chapter, pageInfo);
    }

    Canvas canvas = new Canvas();

    foreach (WordInfo word in pageInfo.Words)
    {
        TextBlock txtblk = new TextBlock
        {
            FontFamily = fontMetrics.Font.FontFamily,
            FontSize = this.fontSize,
            Text = paragraphs[word.ParagraphIndex].Text.
                Substring(word.CharacterIndex,
                word.CharacterCount),
            Tag = word
        };

        Canvas.SetLeft(txtblk, word.LocationLeft);
        Canvas.SetTop(txtblk, word.LocationTop);
        canvas.Children.Add(txtblk);
    }
    return canvas;
}
```

The second reason involves the problem of selecting text. You might want to allow the user to select text for different purposes: perhaps to add notes or annotations to a document, or to look up words or phrases in a dictionary or Bing or Wikipedia, or to simply copy text to the clipboard. You'll need to provide the user with some way to select the text and to display this selected text in a different color.

Can a single TextBlock display different pieces of text in different colors? Yes, that's possible with the Inlines property and a separate Run object for the selected text. It's messy, but it's possible.

The actual pagination and layout
code doesn't touch the UI.

The more difficult problem is letting the user select the text to begin with. The user should be able to click or touch a particular word and then drag to select multiple words. But if an entire paragraph is displayed by a single TextBlock element, how do you know what word that is? You can perform hit-testing on the TextBlock itself, but not on the individual Run objects.

When each word is its own TextBlock, the hit-testing job becomes much easier. Of course, other challenges arise on the phone. Chunky fingers must select tiny text, which means it's probably necessary for the user to enlarge the text before beginning selection.

As usual, as each new feature in a program is introduced, it suggests even more features. ■

CHARLES PETZOLD is a longtime contributing editor to MSDN Magazine. His recent book, "Programming Windows Phone 7" (Microsoft Press, 2010), is available as a free download at bit.ly/cpebookpdf.

THANKS to the following technical expert for reviewing this article:
Chipalo Street



The Power of the Default

Remember Clippy, the dancing, infuriating paper clip that used to pop up in Office? I'll wait while your blood pressure recedes after the surge of hatred that name triggers. Lawyers Dahlia Lithwick and Brandt Goldstein described him well in their book "Me v. Everybody: Absurd Contracts for an Absurd World" (Workman Publishing Company, 2003): "The Maniacal-Paper-Clip-with-Eyebrows Provision. You will delete/destroy/disable whatever it is that allows that inane little [out-of-wedlock child] to leap around the bottom right-hand corner of my screen ... observing: 'I see you're writing a ransom note ...' or assuming that I wish it to turn all my letters into spreadsheets and my correspondence into numbered lists."

And remember the joy at Clippy's demise, with Microsoft Office XP in 2001? Bill Gates got a standing ovation for saying that "XP stands for Ex-Paperclip." Clippy tried to interrupt and was yanked off the stage by a magnet while the crowd cheered. My favorite spoof is from NPR's "Wait, Wait, Don't Tell Me" in which Clippy is dragged into the forest for a Mafia-style execution (bit.ly/kkpAiW): "It looks like you're digging a grave. Can I help dig? Is this a business grave or a personal grave?"

You know what? All that fuss, the cheers and wailing and gnashing of teeth, was about the default state of a single checkbox. A user could always turn Clippy off, either by right-clicking or using the Tools | Options dialog. And after Clippy's publicized removal, he still lurked in Office for six more years, undead, waiting to annoy anyone foolish enough to check the box that would reactivate him.

Behold the power of the default setting. It not only determines a user's vital first impression of your program, it also determines the overall satisfaction of most of your users, and hence the success or failure of your product.

Few users ever change a program's default settings. Some don't know that they can, others don't know where to start. The rest consider it more trouble than it's worth, or fear damaging a working installation. UI guru Alan Cooper considers changing default settings to be the defining characteristic of an advanced user. Thinking of all the applications I use regularly, there isn't one on which I'd consider myself an advanced user in this sense.

"Baloney, Plattska," I hear you saying. "I change my programs' settings all the time, just for the sheer pain of it." Yes you do—because you're a geek. But few of your users ever do.

It's not enough to build a program with the correct feature set, or even with the correct configuration points. You also need to put that set into its optimal configuration by default, so users can use



"I'm Clippy, and I'm here to annoy you!"

it without thinking. That means that you have to know who your users are, because they sure as heck aren't you. (Where have I heard that before?)

For example, the default toolbar in Office 2003 contained a quick print button. Instead of displaying the full print dialog, this button simply printed one copy of the whole document on the default printer. But in Office 2007, the default quick access toolbar (far upper left) doesn't contain a quick print button, although it contains Save, Undo and Redo. You'd think that quick printing is a feature that most users want; therefore it should be turned on by default.

To make this decision correctly, you need data about your users—not guesses, not "you'd think," but good, hard data. My UI consulting clients sometimes object to the time and cost of data gathering, but there's no substitute for it. If the Office team has data from their Customer Experience Improvement Program that proves most users don't care about quick printing, then I withdraw my objection to its absence.

It's not enough to build a program with the correct feature set, or even with the correct configuration points. You also need to put that set into its optimal configuration by default.

I'll leave you with a scary thought. Clippy is actually back now, in the training game Ribbon Hero (bit.ly/mC1x0s). Imagine the horrors if he ever metastasized to Windows Phone 7. "I see you're in a bar and you're calling your ex. Are you sure that's a good idea?" At least turn *him* off by default. ■

DAVID S. PLATT teaches Programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.

Native Microsoft® Excel® Support

GrapeVine TV Presents

YOUR ONLINE SOURCE FOR DEVELOPER NEWS

*Live from GvTv,
be a part of the launch!*



GvTv.GCPowerTools.com

Smarter Components for Smarter Developers

Excellent

EXPORT

Printing Printing to PDF

Reporting

Multiple headers

Borders

Merging

Spanning

Conditional formatting



GrapeCity® PowerTools

www.GCPowerTools.com

Hundreds of Chart styles for data visualization

Expertise



The Dashboard Framework for Developers like you

Support For Numerous Data Sources

Extensible And Customizable With Our Open API

Rapid Dashboard Development

Dundas Communication (Annotations)

Full scripting Capabilities

Extend The MS BI Stack And SharePoint

Notifications/Alerts

V2.5 Now Available

Dundas Dashboard was built with developers and IT staff in mind. Whether it's our open API, simple web integration or powerful scripting capabilities, technologists have all the tools and options they need for getting their dashboard projects up and running quickly and easily.



Powered by
Microsoft Silverlight™



www.dundas.com
(416) 467-5100 • (800) 463-1492

Silverlight is a trademark of Microsoft Corporation in the United States and/or other countries.