



magazine
msdn®

ASYNCHRONOUS PROGRAMMING

Easier Asynchronous Programming with the New Visual Studio Async CTP

Eric Lippert..... 22

Pause and Play with Await

Mads Torgersen 28

Async Performance: Understanding the Costs of Async and Await

Stephen Toub 34

PLUS:

Securing Access to LightSwitch Applications

Valerie Andersen, Matt Evans, Sheel Shah and Michael Simons 46

Authoring an F#/C# VSIX Project Template

Dan Mohl..... 58

Harnessing the Power of the Dynamics CRM 4.0 API from Silverlight 4

Mark Beckner 66

Building Information Architecture in SharePoint 2010

Shahram Khosravi 70

COLUMNS

CUTTING EDGE

Objects and the Art
of Data Modeling
Dino Esposito page 6

WINDOWS WITH C++

Thread Pool Cancellation
and Cleanup
Kenny Kerr page 12

FORECAST: CLOUDY

The Windows Azure
AppFabric Service Bus: Topics
Joseph Fultz page 16

TEST RUN

Graph Structures
and Maximum Clique
James McCaffrey page 82

UI FRONTIERS

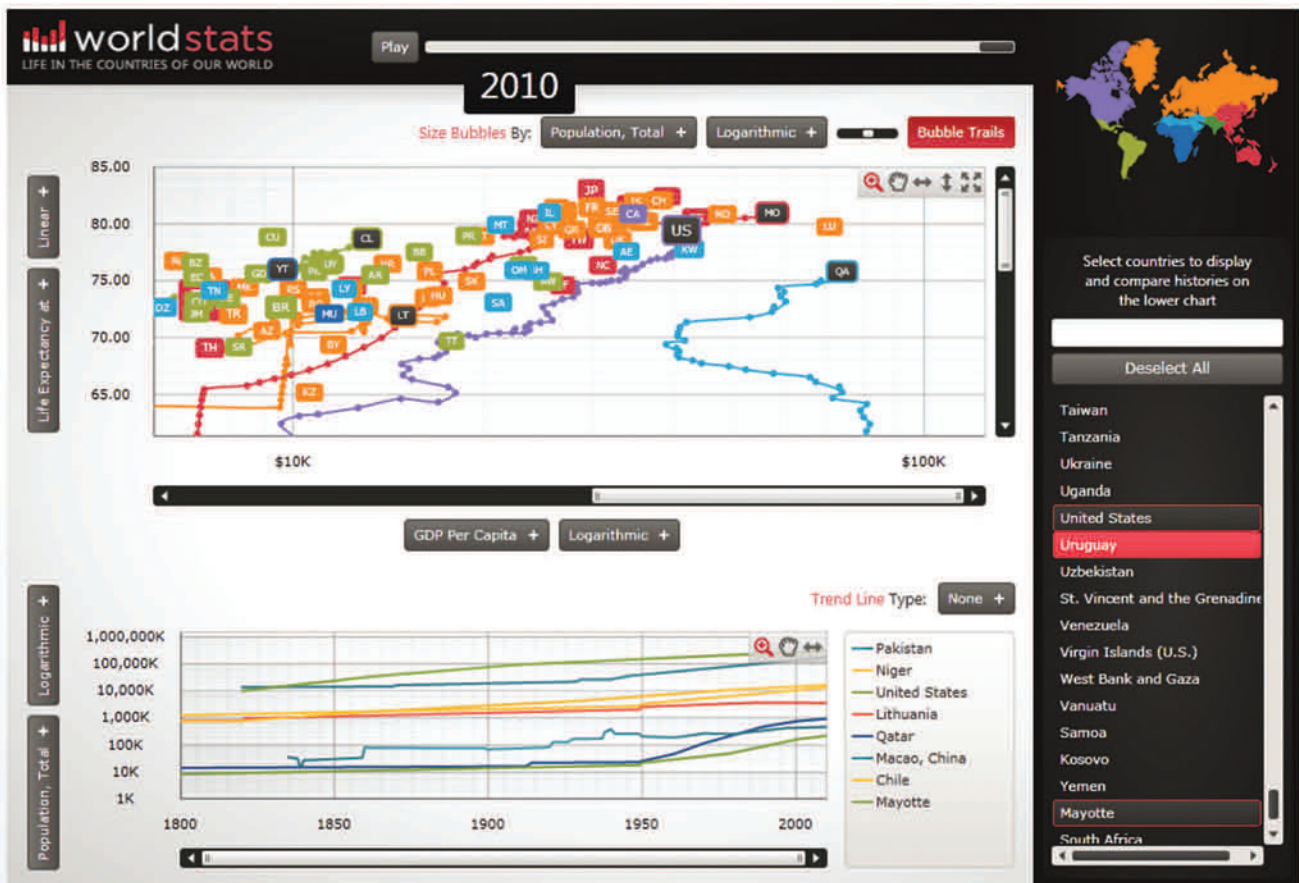
Pages and Pop-ups
in Windows Phone 7
Charles Petzold page 90

DON'T GET ME STARTED

Imagine That
David Platt page 96

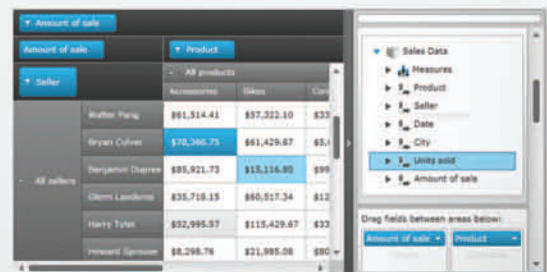
NetAdvantage® ULTIMATE

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



INFRAGISTICS 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.



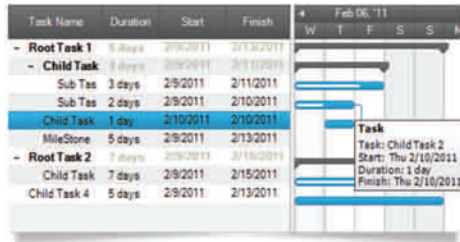
OLAP PIVOT GRID DATA VISUALIZATION

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.



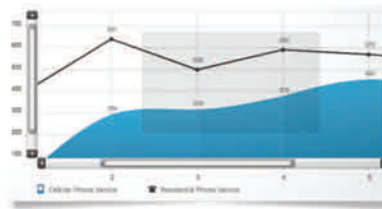
WINDOWS FORMS GANTT CHARTS

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		
Equals	Adjustable Race	AR-5381
Does not equal	Bearing Ball	BA-8327
Greater than	3B Ball Bearing	BE-2349
Less than	Headset Ball Bearings	BE-2908
Greater than or equal to	Blade	BL-2036
Less than or equal to	LL Crankarm	CA-5965
318	ML Crankarm	CA-6738
319	HL Crankarm	CA-7457
320	Chainring Bolts	CB-2903
321	Chainring Nut	CN-6137

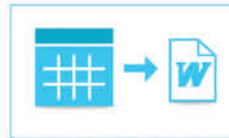
JQUERY

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



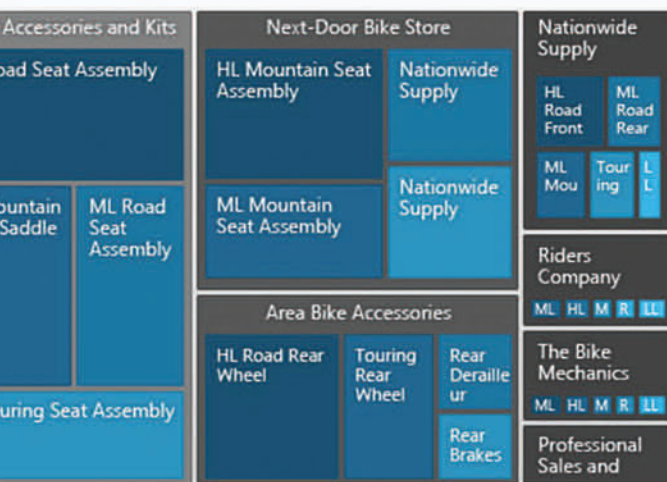
CHARTING

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

Web with Spider

Network with Spider

Engine for Win & .NET

Publish (portable media)

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

OCTOBER 2011 VOLUME 26 NUMBER 10

LUCINDA ROWLEY Director

KIT GEORGE Editorial Director/mmeditor@microsoft.com

PATRICK O'NEILL Site Manager

MICHAEL DESMOND 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. Periodicals postage paid at 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 U.S. 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: PO, Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. 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



Embarcadero RAD Studio XE2

The ultimate application development suite for Windows, Mac, mobile and Web

by Embarcadero

Embarcadero® RAD Studio XE2 is the ultimate application development suite and the fastest way to build data-rich, visually engaging applications for Windows, Mac, mobile, .NET, PHP and the Web. RAD Studio includes Delphi®, C++Builder® and RadPHPTM, enabling developers to deliver applications up to 5x faster across multiple desktop, mobile, Web, and database platforms including Delphi applications for 64-bit Windows.

NEW VERSION!

Professional Ed.
TechXtend #
CGI 15501A01

\$1,383.99

techxtend.com/embarcadero

UltraEdit

The world's #1 text editing solution is also the world's most affordable!

by IDM Computer Solutions

UltraEdit is the world's standard in text editors. Millions use UltraEdit as the ideal text/hex/programmers editor on any platform — Windows, Mac, or Linux!

Features include syntax highlighting for nearly any programming language; powerful Find, Replace, Find in Files, and Replace in Files; FIP support, sort, column mode, hex, macros/scripting, large file handling (4+ GB), projects, templates, Unicode, and more.



Named User
1-24 Users
TechXtend #
184 01201A01

\$59.95

techxtend.com/idm

VMware vSphere 5 Essentials Kit Bundle

VMware vSphere is the industry-leading virtualization platform for building cloud infrastructures that enables users to run business critical applications with confidence and respond faster to their business.

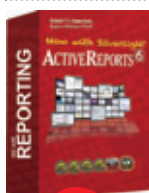
vSphere accelerates the shift to cloud computing for existing datacenters, while also underpinning compatible public cloud offerings paving the way for the only hybrid cloud model. With over 250,000 customers worldwide and the support of over 2500 applications from more than 1400 ISV partners, VMware vSphere is the trusted platform for any application.



NEW VERSION 5!

CALL

techxtend.com/vSphere



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

NEW VERSION 6!

Professional Ed.
TechXtend #
D03 04301A01

\$1,310.99

techxtend.com/grapecity

Zemana Antilogger

by Zemana LLC

Stop hackers now! Zemana AntiLogger has a unique technology that detects when malware runs on your computer, and shuts it down — before it can steal your identity or hurt your computer. It doesn't slow your computer down like those old-fashioned antivirus programs that rely on virus signature updates and file scanning in order to prevent malware attacks. Zemana AntiLogger eliminates threats from keyloggers, SSL banker trojans, spyware, and more! Plus, you can use Zemana AntiLogger seamlessly, as an important extra layer of security, in harmony with almost any antivirus or firewall software.



TechXtend #
ZOB 01101G01

\$10.42

techxtend.com/zemana

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

New Version Released!



Professional Edition
TechXtend #
T79 12101A01

\$1,109.99

Download a demo today.

techxtend.com/textcontrol



Microsoft Visual Studio Professional 2010

by Microsoft

Microsoft Visual Studio 2010 Professional with MSDN Essentials Subscription is an integrated environment that simplifies creating, debugging and deploying applications. Unleash your creativity and bring your vision to life with powerful design surfaces and innovative collaboration methods for developers and designers. Work within a personalized environment, targeting a growing number of platforms, including Microsoft SharePoint and cloud applications and accelerate the coding process by using your existing skills.

Upgrade
TechXtend #
M47 40201B02

\$478.99

techxtend.com/microsoft

Lenovo ThinkPad X220

by Lenovo

The ThinkPad X220 features the quality and performance Lenovo 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. ThinkPad has improved on full-size keyboard design and nimble TrackPoint by adding an advanced buttonless touchpad.



TechXtend #
ZHI GF0818

\$1,362.99

techxtend.com/lenovo

Kingston DataTraveler G3 16 GB Flash Drive

by Kingston Technology

The new generation of a Kingston best-seller is here! With capacities up to 32GB, the reliable DataTraveler Generation 3 (G3) is ideal for your important documents, music, video clips and favorite photos that can be stored and retrieved in a flash.

Available in four fun colors by capacity, it's a perfect fit for the office, home, school and wherever you travel. A well-built cap protects the USB plug and your data. DataTraveler G3 also makes a great promotional item for your organization.



TechXtend #
ZHI DQ1077

\$28.99

techxtend.com/kingston



VMware Workstation 7 for Linux & Windows

The world's #1 text editing solution is also the world's most affordable!

by VMware

Winner of more than 50 industry awards, VMware Workstation transforms the way technical professionals develop, test, demo, and deploy software. Innovative features help software developers, QA engineers, sales professionals, and IT administrators to reduce hardware cost, save time, minimize risk, and streamline tasks that save time and improve productivity.

TechXtend #
V55 22301A04

\$149.99

techxtend.com/vmware

Programmer's Paradise has 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 **WEBTRW10** 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.



Thinkin' About Async

This month *MSDN Magazine* explores the powerful new asynchronous programming features coming to the next versions of C#, Visual Basic and the Microsoft .NET Framework. Available today via the Visual Studio Async CTP, the new functionality helps resolve one of the biggest emerging headaches in the age of multicore systems and cloud computing: the penalties incurred when threads are blocked waiting for things like data to return from a remote server or a computation on another thread to complete.

I spoke with Mads Torgersen, author of one of this month's features on asynchronous programming and a principal program manager on the C# and Visual Basic Language team at Microsoft. He notes that almost every application is becoming connected and that latency is emerging as "a central problem to code around." So why didn't we see async support in the .NET Framework 4?

"At the point where the feature set was locked down for that release, we didn't have a serious grasp on what such a language feature should look like," says Torgersen. "However, F# shipped at the same time with an asynchronous language feature that ended up inspiring us a lot. By the time we shipped C# 4 and Visual Basic 10, we were seeing mounting evidence that asynchrony should be next in line, and that it was a problem that really needed to be solved at the language level."

Torgersen describes the core implementation approach as quite simple. The solution was to "pause and resume" the execution of code in flight. Rather than chop up generated code into separate bits, the team used a technique to "parachute in" to the proper spot in the user's code. The approach provides the benefit of retaining the structure of the original source code—and the results are hard to argue with.

"We've had a surprisingly small bug tail of inconsistencies," Torgersen says.

Async Evolution

The new asynchronous programming features in C# and Visual Basic mark the latest in a series of important evolutionary steps for Microsoft's flagship managed programming languages. Over the past few years, the languages have taken on aspects of functional and

dynamic programming, and most recently asynchronous programming. As Lisa Feigenbaum, program manager in the Visual Studio group at Microsoft, explains, the effort was made to incorporate these capabilities in ways that "best fit the style of each language."

"For example, dynamic language interoperability was added to Visual Basic by taking advantage of the late binding constructs that were already part of the language. Furthermore, in C# we added a static type, called *dynamic*," Feigenbaum says. "Speaking of functional programming, when adding lambdas to Visual Basic, we used the familiar keyword syntax with *Sub/End Sub* and *Function/End Function*. However, for C#, which is less verbose, we used symbols to design the syntax with *'=>'*. These designs help each language preserve its original character and remain familiar to developers as it grows."

Adding significant new functionality to a programming language is never a trivial matter. Feigenbaum describes language design as a "very conservative process," emphasizing that features are not added until they're exactly right. In the case of async, the new functionality is built on foundations introduced with the last version of the .NET Framework.

"For example, the Visual Studio vNext async designs build upon the *Task* type that was added to the .NET Framework in .NET 4," says Feigenbaum. "Using that foundation, as well as additional Framework support that's being added in the next version, we were ultimately able to design a very elegant language syntax that we were happy with for async in Visual Studio vNext."

Today, developers can work with the Async CTP to become familiar with the new resources coming to C# and Visual Basic. In the meantime, Torgersen says developers can do one thing to best prepare for the age of async: become *Task*-based.

"Even on .NET 4, the *Task* types are a much better currency for asynchronous activity than the older patterns we have. You still need to be callback-based because you don't have the language support, but in a more elegant way. And having all your signatures follow the new *Task*-based pattern will prepare you beautifully for the day when you can con-

sume those with just a simple 'await' expression."

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.

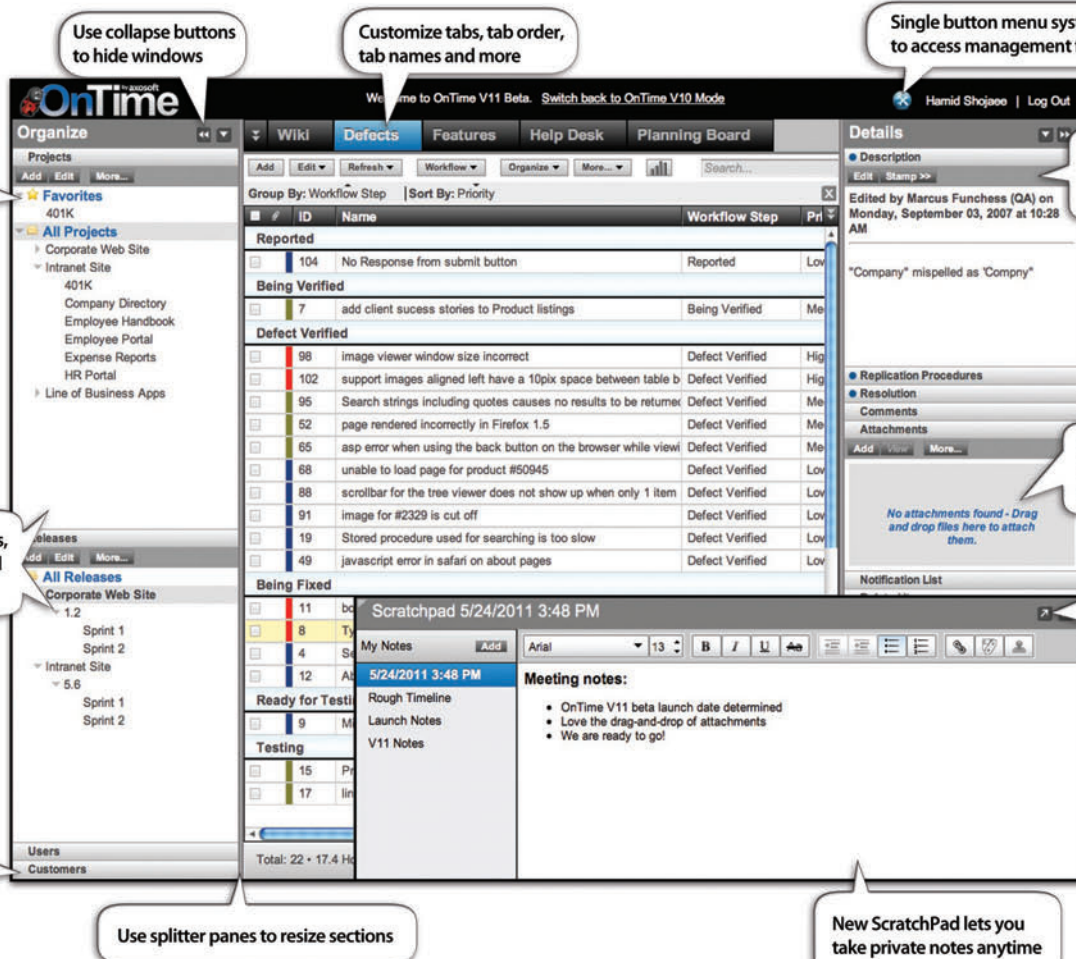
Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly 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.

Everything You Need to Ship Software OnTime

Agile Project Management • Bug Tracking • Backlogs • Burndowns • All in a Beautiful UI



Use collapse buttons to hide windows

Customize tabs, tab order, tab names and more

Single button menu system to access management features

Add your most used projects to Favorites

See Projects, Releases, Users & Customers all at the same time

Collapse / remove sections you don't want right now

See any or all the details of an item with a glance

Drag-and-drop files to attach them to the item

Use the "Throw" button to open in a new window

Use splitter panes to resize sections

New ScratchPad lets you take private notes anytime

Visit www.axosoft.com for the following:



FREE FOR 2 Users

Never expires, no credit card required. 2 users are totally free forever. Install on your servers or sign up for an account.



Making of OnTime 11

Watch the key principles, the design philosophy and what separates OnTime from other competing products.



Join a Live Demo

Refreshingly informative, and certainly not a sales pitch. We offer 3 live demos a week, so reserve your spot today!



Objects and the Art of Data Modeling

Many of today's apps are built around a single data model, typically persisted to a data store via an object-relational mapper (ORM) tool. But sometimes—for several different reasons—you may need more flexibility, which requires multiple models. In this article, I'll discuss some strategies you can use to handle these situations and develop more layered and robust applications.

Using different models clearly makes the whole application more complex, but it's a sort of necessary, positive complexity that makes the whole project more manageable. Understanding when one model of data just doesn't fit all use cases is the challenge for the architect.

Different parts of a software application may have their own model of data. The way in which you represent data at the UI level is different from the way in which you organize data in the middle tier, and it may be even different from how the data is physically persisted in some data store.

But, for many years, developers used just one model of data, regardless of the part of the application involved. Many of us have grown with relational databases and their related modeling techniques. It was natural to spend a lot of effort in elaborating a model based on relations and normalization rules. Any data stored in a relational database was then extracted and moved around using memory structures similar to an in-memory database. Record Set is the name of this generic data pattern (see bit.ly/nQnyaf); the ADO.NET DataSet was an excellent implementation of it.

The table-based approach was progressively pushed to the corner by the growing complexity of applications. ORM tools emerged for two main pragmatic reasons. First, working with objects is easier than dealing with generic super-arrays of data such as a recordset. The second reason is productivity. The goal of an ORM is taking an object model and mapping it to a relational schema. By using an ORM, you essentially make the object model look like the real database to the application's eyes.

How do you design this model? And should you really use just one model for each application? Let's find out.

One Model Doesn't Always Fit All

The recent release of the Entity Framework (EF) 4.1 included support for "Code First" programming, which, as the name implies, lets Microsoft .NET Framework developers take a code-first approach to data modeling. This basically means that you begin by writing the Plain Old CLR Object (POCO) classes that model the domain of the application. The second step consists of mapping these classes to some persistent store and having the ORM tool (the EF) take care

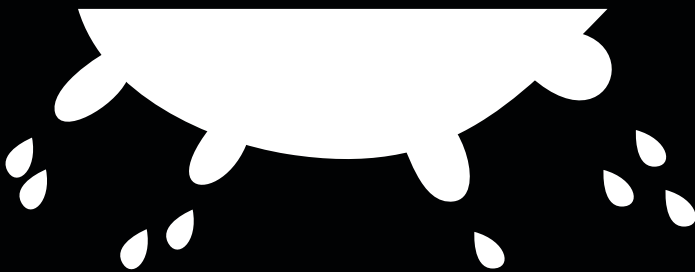
of persistence details. The ORM exposes a query language (LINQ to Entities), transactional semantics (the `xxxContext` objects in the EF) and a basic create, read, update and delete (CRUD) API that supports concurrency, lazy loading and fetch plans.

So you have the model, and you know how to persist it and how to query information from it. Is this a model you can use everywhere in your application? More specifically, is this a model you can effectively bring up to the presentation layer? This is a really sore point where theory and practice differ substantially, context is king and uncertainty (when not really confusion) reigns. Let me offer a concrete example bound to the popular ASP.NET MVC technology. The only reason I'm using ASP.NET MVC instead of, say, Silverlight, is that ASP.NET MVC has the word Model in the name (the M in MVC)—and I've been asked too many times in classes and conferences about the exact location of the "model" in an ASP.NET MVC application.

Understanding when one model of data just doesn't fit all use cases is the challenge for the architect.

Today, even after three versions, too many tutorials on ASP.NET MVC insist on using just one model for queries, updates and presentation. Many tutorials also keep the definition of the model inside the main project. It works, so where's the problem? The problem isn't with the solution, which works, is effective, and is something I've often used myself and plan to keep using. There's actually a set of real-world, but simple and relatively short-lived applications (not just demos and tutorials), that can be developed with simple patterns. The problem of using just one model is with the message it transmits to the primary consumers of tutorials: developers looking to learn a technology. Having just one model in just one place suggests it's the preferred (if not recommended) way of doing things. It is, instead, just a special case—a very simple and favorable scenario. If the real scenario you'll actually work in matches that, you're more than fine. Otherwise, you're stuck and ready to grow your first "big ball of mud" bigger and bigger.

A DUTCH COW IS MILKED 10x FASTER



Telerik RadControls for ASP.NET AJAX

helped redesign the website of one of the largest Dutch yogurt producers in just two months.

The revamped site increased user traffic tenfold and became the company's primary consumer communications channel.

Read other customers' stories:
www.telerik.com/happycustomers

 **telerik**
deliver more than expected

A More General Architecture

Let's begin by using slightly different names. Let's call the model of data that you persist the domain model and call the data you manage in the view the view model. I should mention that domain model isn't exactly a neutral term in software, as it refers to an object model designed according to a number of additional rules. In the scope of this article, I'm not using the meaning that results from the Domain-Driven Design (DDD) methodology. For me, here, the domain model is simply the object model you persist—*entity model* might be another equivalent—and less confusing—term.

You use the classes in the entity model in the back end of the application; you use the classes in the view model in the presentation layer. Note, however, that in ASP.NET MVC, the presentation layer is the controller. The controller should receive data ready for the UI. The middle-tier components receive and return view model objects and internally use entity objects. **Figure 1** shows the web of dependencies in a typical multilayered project.

The presentation (that is, the codebehind or controller class) references the application logic, namely a component that implements the application's use cases. The application logic technically belongs to the business layer or tier, and in very simple cases may be merged with the presentation. This is what happens in some tutorials that are either too simple to really feel the need for isolating application logic in its own layer, or poorly designed and splitting application logic between the presentation and the data access layer (DAL).

The application logic assembly implements the service layer pattern and decouples two interfacing layers: presentation and data. The application logic references both the entity model (your domain classes) and the DAL. The application logic orchestrates the DAL, domain classes and services to pursue the expected behavior. The application logic communicates with the presentation layer through view model objects and communicates with the DAL through domain objects. The DAL, in turn, references the model and the ORM assembly.

A Few Words on the Entity Framework

Let's examine this architecture, assuming the EF as the ORM. The EF isn't simply an ORM, but as its name suggests, it does the typical work of an ORM, plus offering a framework for creating a model. Nice idea, but it shouldn't be forgotten that we're straddling two distinct layers—business and the DAL. The classes are business; the persistence engine is the DAL. For the EF, the persistence engine (the ORM assembly) is `system.data.entity` and its dependencies, including the `ObjectContext` class. Put another way, unless you use POCO classes and Code First, you'll likely end up with a domain model

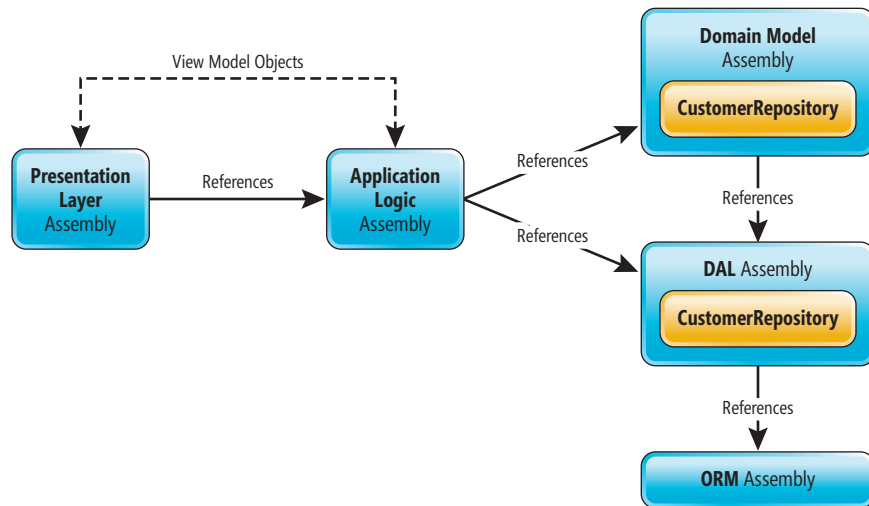


Figure 1 Connections Among Layers

that has a dependency on the ORM. A domain model should use POCO—that is, among other things, it should be a self-contained assembly. An interesting thread on this topic exists on Stack Overflow at bit.ly/mUs6cv. For more details on how to split entities from context in the EF, see the ADO.NET team blog entries, “Walkthrough: POCO Template for the Entity Framework” (bit.ly/bDcUoN) and “EF 4.1 Model & Database First Walkthrough” (bit.ly/hufcWN).

From this analysis, it seems that the POCO attribute is essential for domain classes. POCO, of course, indicates a class with no dependencies outside its own assembly. POCO is about simplicity and is never wrong. Not going the POCO way means having forms of tight coupling between layers. Tight coupling isn't poisonous and doesn't kill you instantly, but it can take your project to a slow death. Check out my column about software disasters in last month's issue (msdn.microsoft.com/magazine/hh394145).

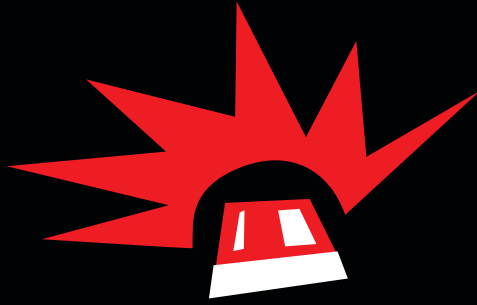
A domain model should use POCO—that is, among other things, it should be a self-contained assembly.

What's the Model?

When you create an object model, you create a library of classes. You can organize your object model according to a number of patterns, but essentially it boils down to choosing between a table-oriented approach and an object-oriented approach.

How do you devise your classes? Which capabilities should they feature? In particular, should your classes be aware of the database? Should they include logic (that is, methods) or be limited to just exposing properties? There are two main patterns you can refer to: Active Record and Domain Model.

In the Active Record pattern, your classes are closely modeled after database tables. You mostly have one class per table and one



MOUTH-TO-MOUTH ON A MASS SCALE



Telerik RadControls for Windows Phone

helped cut development time building a WP7 app to assist non-medical personnel in emergency situations.

Using Telerik controls allowed the development team to concentrate their efforts on building the app's core and contents. It is now one of the top applications in its category.

Read other customers' stories:
www.telerik.com/happycustomers

 **telerik**
deliver more than expected

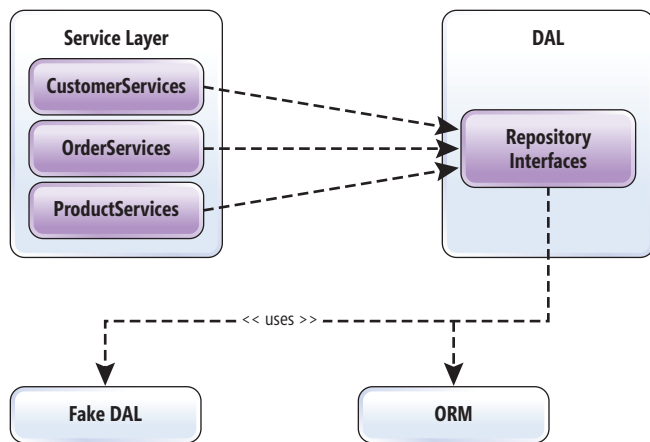


Figure 2 Using Repositories in the DAL

property per column. More importantly, classes are responsible for their own persistence and their own simple, minimal domain logic.

According to the Domain Model pattern, your classes are aimed at providing a conceptual view of the problem's domain. These classes have no relationships with the database and have both properties and methods. Finally, these classes aren't responsible for their own persistence. If you opt for a Domain Model approach, persistence has to be delegated to a distinct layer—the DAL. You can write this layer yourself, but it wouldn't be much fun. A well-done DAL for a library designed according to the Domain Model pattern is nearly the same as an ORM tool. So why not use one of the existing ORM tools?

To improve the design, you can establish a dependency between application logic and the DAL through repository interfaces.

Especially if you use the automatic code generator, the EF gets you an object model made of classes that have only properties. The pattern used certainly isn't Active Record, but classes have no methods by default. Fortunately, classes are marked as partial, which makes it possible for you to shape up a much richer domain model by adding logic through partial classes. If you choose the Code First approach, you're then entirely responsible for writing the source code of your classes from start to finish.

Classes in an object model that only feature properties are often referred to as an anemic domain model.

Repositories

A good practice that's gaining well-deserved popularity is wrapping data access code in façade classes known as repositories. A repository consists of an interface and an implementation class. You typically have a repository for each *significant* class in the model. A significant class in an object model is a class that controls

its own persistence and stands on its own in the domain without depending on other classes. In general, for example, Customer is a significant class, whereas OrderDetails is not, because you'll always use order details if you have an order. In DDD, a significant class is said to be an aggregate root.

To improve the design, you can establish a dependency between application logic and the DAL through repository interfaces. Actual repositories can then be injected in the application logic as appropriate. **Figure 2** shows the resulting architecture where the DAL is based on repositories.

Repositories enable dependency injection in the DAL because you can easily unplug the current module that provides persistence logic and replace it with your own. This is certainly beneficial for testability, but it isn't limited to that. An architecture based on repositories allows mocking the DAL and testing the application logic in isolation.

Repositories also represent an excellent extensibility point for your applications. By replacing the repository, you can replace the persistence engine in a way that's transparent to the rest of the layers. The point here isn't so much about switching from, say, Oracle to SQL Server, because this level of flexibility is already provided by ORM tools. The point here is being able to switch from the current implementation of the DAL to something different such as a cloud API, Dynamics CRM, a NoSQL solution and the like.

What Repositories Should Not Be

A repository is part of the DAL; as such, it isn't supposed to know about the application logic. This may be too obvious a statement, but a type of architecture diagram I see quite often these days is based on presentation and repositories.

If your actual logic is thin and simple—or is essentially CRUD—then this diagram is more than OK. But what if that's not the case? If so, you definitely need a place in the middle where you deploy your application logic. And, trust me, I've seen only a few applications that are just CRUD in my career. But probably I'm just not a lucky man ...

Clarifying Obscurity

Wrapping up, nowadays applications tend to be designed around a model of data that an ORM tool then persists to some data store. This is fine for the back end of the application, but don't be too surprised if the UI requires you to deal with significantly different aggregates of data that just don't exist in the original model. These new data types that exist for the sole purpose of the UI must be created. And they end up forming an entirely parallel object model: the view model. If you reduce these two models to just one, then by all means stick to that and be happy. If not, hopefully this article clarified some obscure points. ■

DINO ESPOSITO is the author of "Programming Microsoft ASP.NET MVC3" (Microsoft Press, 2011) 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. You can follow him on Twitter at twitter.com/despos.

THANKS to the following technical experts for reviewing this article:
Andrea Saltarello and Diego Vega



A GLOBAL MEETING SHORTAGE IS COMING



Telerik TeamPulse,

an Agile project management solution, helped a global engineering powerhouse with teams and operations dispersed across the globe to reduce their development time by 50 percent.

Introducing TeamPulse allowed the teams to meet their specific goals with far fewer meetings and less documentation.

Read other customers' stories:
www.telerik.com/happycustomers

 **telerik**
deliver more than expected



Thread Pool Cancellation and Cleanup

Cancellation and cleanup are notoriously difficult problems to solve when it comes to multithreaded applications. When is it safe to close a handle? Does it matter which thread cancels an operation? To make matters worse, some multithreaded APIs are not reentrant, potentially improving performance but also adding complexity for the developer.

I introduced the thread pool environment in last month's column (msdn.microsoft.com/magazine/hh394144). One critical feature this environment enables is cleanup groups, and that's what I'll focus on here. Cleanup groups don't attempt to solve all of the world's cancellation and cleanup problems. What they do is make the thread pool's objects and callbacks more manageable, and this can indirectly help to simplify the cancellation and cleanup of other APIs and resources as needed.

So far, I've only shown you how to use the `unique_handle` class template to automatically close work objects via the `CloseThreadpoolWork` function. (See the August 2011 column at msdn.microsoft.com/magazine/hh335066 for the details.) There are some limitations with this approach, however. If you want a say in whether pending callbacks are canceled or not, you need to call `WaitForThreadpoolWorkCallbacks` first. That makes two function calls multiplied by the number of callback-generating objects in use by your application. If you opt to use `TrySubmitThreadpoolCallback`, you don't even get the opportunity to do so and are left wondering how to cancel or wait for the resulting callback. Of course, a real-world application will likely have far more than just work objects. In next month's column, I'll start introducing the other thread pool objects that produce callbacks, from timers to I/O to waitable objects. Coordinating the cancellation and cleanup of all of these can quickly become a nightmare. Fortunately, cleanup groups solve these problems and more.

The `CreateThreadpoolCleanupGroup` function creates a cleanup group object. If the function succeeds, it returns an opaque pointer representing the cleanup group object. If it fails, it returns a null pointer value and provides more information via the `GetLastError` function. Given a cleanup group object, the `CloseThreadpoolCleanupGroup` function instructs the thread pool that the object may be released. I've mentioned this before in passing, but it bears repeating—the thread pool API does not tolerate invalid arguments. Calling `CloseThreadpoolCleanupGroup` or any of the other API functions with an invalid, previously closed or null pointer value will cause your application to crash. These are defects introduced by the programmer and should not require additional checks at

run time. The `unique_handle` class template I introduced in my July 2011 column (msdn.microsoft.com/magazine/hh288076) takes care of these details with the help of a cleanup-group-specific traits class:

```
struct cleanup_group_traits
{
    static PTP_CLEANUP_GROUP invalid() throw()
    {
        return nullptr;
    }

    static void close(PTP_CLEANUP_GROUP value) throw()
    {
        CloseThreadpoolCleanupGroup(value);
    }
};
```

```
typedef unique_handle<PTP_CLEANUP_GROUP, cleanup_group_traits> cleanup_group;
```

I can now use the convenient typedef and create a cleanup group object as follows:

```
cleanup_group cg(CreateThreadpoolCleanupGroup());
check_bool(cg);
```

As with private pools and callback priorities, a cleanup group is associated with various callback-generating objects by means of an environment object. First, update the environment to indicate the cleanup group that will manage the lifetime of objects and their callbacks, like this:

```
environment e;
SetThreadpoolCallbackCleanupGroup(e.get(), cg.get(), nullptr);
```

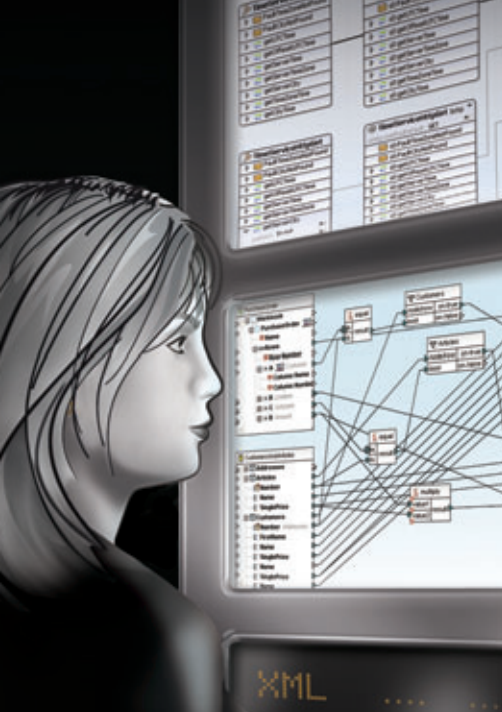
At this point, you can add objects to the cleanup group that are then referred to as members of the cleanup group. These objects can also be individually removed from the cleanup group, but it's more common to close all members in a single operation.

A work object can become a member of a cleanup group at creation time simply by providing the updated environment to the `CreateThreadpoolWork` function:

```
auto w = CreateThreadpoolWork(work_callback, nullptr, e.get());
check_bool(nullptr != w);
```

Notice that I didn't use a `unique_handle` this time. The newly created work object is now a member of the environment's cleanup group and its lifetime need not be tracked directly using RAII.

You can revoke the work object's membership only by closing it, which can be done on an individual basis with the `CloseThreadpoolWork` function. The thread pool knows that the work object is a member of the cleanup group and revokes its membership before closing it. This ensures that the application doesn't crash when the cleanup group later attempts to close all of its members. The inverse isn't true: If you first instruct the cleanup group to close all of its members and then call `CloseThreadpoolWork` on the now invalid work object, your application will crash.



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



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

NEW in Version 2011:

- Instant chart generation to analyze XML and database data
- StyleVision® integration in MapForce for report creation
- XML digital signatures
- Data streaming for ETL
- Support for IATA PADIS & HIPAA EDI
- SOAP validation against WSDL

The Altova MissionKit includes multiple intelligent tools for data integration:

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

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

XMLSpy® – XML editor and Web services tool

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

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

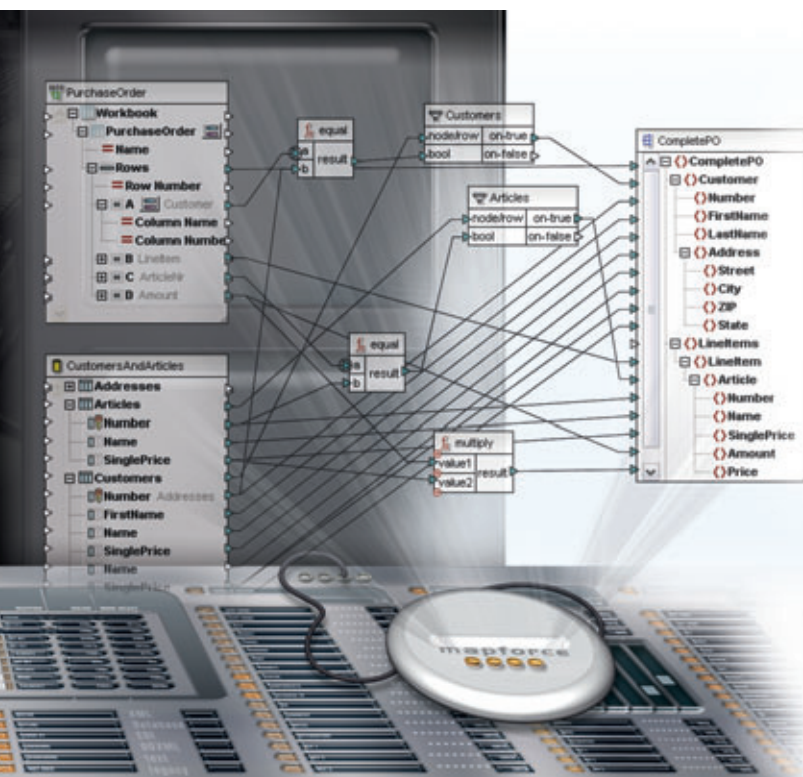
- Support for all major relational databases and translation between DB types
- SQL editor, graphical database design & content editor

 **Download a 30 day free trial!**

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



Scan with your smart-phone to learn more about these products



Of course, the whole point of a cleanup group is to free the application from having to individually close all of the various callback-generating objects it happens to be using. More importantly, it allows the application to wait for and optionally cancel any outstanding callbacks in a single wait operation rather than having to have an application thread wait and resume repeatedly. The `CloseThreadPoolCleanupGroupMembers` function provides all of these services and more:

```
bool cancel = ...
CloseThreadPoolCleanupGroupMembers(cg.get(), cancel, nullptr);
```

This function might appear simple, but in reality it performs a number of important duties as an aggregate on all of its members. First, depending on the value of the second parameter, it cancels any pending callbacks that haven't yet begun to execute. Next, it waits for any callbacks that have already begun to execute and, optionally, any pending callbacks if you chose not to cancel them. Finally, it closes all of its member objects.

Some have likened cleanup groups to garbage collection, but I think this is a misleading metaphor. If anything, a cleanup group is more like a Standard Template Library (STL) container of callback-generating objects. Objects added to a group will not be automatically closed for any reason. If you fail to call `CloseThreadPoolCleanupGroupMembers`, your application will leak memory. Calling `CloseThreadPoolCleanupGroup` to close the group itself won't help either. You should instead just think of a cleanup group as a way of managing the lifetime and concurrency of a group of objects. You can, of course, create multiple cleanup groups in your application to manage different groups of objects individually. It's an incredibly useful abstraction—but it's not magic, and care must be taken to use it correctly. Consider the following faulty pseudo-code:

```
environment e;
SetThreadPoolCallbackCleanupGroup(e.get(), cg.get(), nullptr);

while (app is running)
{
    SubmitThreadPoolWork(CreateThreadPoolWork(work_callback, nullptr, e.get()));

    // Rest of application.
}

CloseThreadPoolCleanupGroupMembers(cg.get(), true, nullptr);
```

Predictably, this code will use an unbounded amount of memory and will get slower and slower as system resources are exhausted.

In my August 2011 column, I demonstrated that the seemingly simple `TrySubmitThreadPoolCallback` function is rather problematic because there's no simple way to wait for its callback to complete. This is because the work object is not actually exposed by the API. The thread pool itself, however, suffers no such restriction. Because `TrySubmitThreadPoolCallback` accepts a pointer to an environment, you can indirectly make the resulting work object a member of a cleanup group. In this way, you can use `CloseThreadPoolCleanupGroupMembers` to wait for or cancel the resulting callback. Consider the following improved pseudo-code:

```
environment e;
SetThreadPoolCallbackCleanupGroup(e.get(), cg.get(), nullptr);

while (app is running)
{
    TrySubmitThreadPoolCallback(simple_callback, nullptr, e.get());

    // Rest of application.
}

CloseThreadPoolCleanupGroupMembers(cg.get(), true, nullptr);
```

I could almost forgive a developer for thinking this is akin to garbage collection, because the thread pool automatically closes the work object created by `TrySubmitThreadPoolCallback`. Of course, this has nothing to do with cleanup groups. I described this behavior in my July 2011 column. The `CloseThreadPoolCleanupGroupMembers` function in this case is not initially responsible for closing the work object, but only for waiting for and possibly canceling callbacks. Unlike the previous example, this one will run indefinitely without using undue resources and still provide predictable cancellation and cleanup. With the help of callback groups, `TrySubmitThreadPoolCallback` redeems itself, providing a safe and convenient alternative. In a highly structured application where the same callback is queued repeatedly, it would still be more efficient to reuse an explicit work object, but the convenience of this function can no longer be dismissed.

Very little is certain in multithreaded code.

Cleanup groups provide one final feature to simplify your application's cleanup requirements. Often, it's not enough to simply wait for outstanding callbacks to complete. You might need to perform some cleanup task for each callback-generating object once you're sure no further callbacks will execute. Having a cleanup group manage the lifetime of these objects also means that the thread pool is in the best position to know when such cleanup tasks should happen.

When you associate a cleanup group with an environment through the `SetThreadPoolCallbackCleanupGroup` function, you can also provide a callback to be executed for each member of the cleanup group as part of the `CloseThreadPoolCleanupGroupMembers` function's process of closing these objects. Because this is an attribute of the environment, you can even apply different callbacks to different objects belonging to the same cleanup group. In the following example I create an environment for the cleanup group and cleanup callback:

```
void CALLBACK cleanup_callback(void * context, void * cleanup)
{
    printf("cleanup_callback: context=%s cleanup=%s\n", context, cleanup);
}
```

```
environment e;
SetThreadPoolCallbackCleanupGroup(e.get(), cg.get(), cleanup_callback);
```

The cleanup callback's first parameter is the context value for the callback-generating object. This is the context value you specify when calling the `CreateThreadPoolWork` or `TrySubmitThreadPoolCallback` functions, for example, and it's how you know which object the cleanup callback is being called for. The cleanup callback's second parameter is the value provided as the last parameter when calling the `CloseThreadPoolCleanupGroupMembers` function.

Now consider the following work objects and callbacks:

```
void CALLBACK work_callback(PTP_CALLBACK_INSTANCE, void * context, PTP_WORK)
{
    printf("work_callback: context=%s\n", context);
}
```

```
void CALLBACK simple_callback(PTP_CALLBACK_INSTANCE, void * context)
{
    printf("simple_callback: context=%s\n", context);
}
```



```
}
```

```
SubmitThreadPoolWork(CreateThreadPoolWork(work_callback, "Cheetah", e.get()));  
SubmitThreadPoolWork(CreateThreadPoolWork(work_callback, "Leopard", e.get()));  
check_bool(TrySubmitThreadPoolCallback(simple_callback, "Meerkat", e.get()));
```

Which of these is not like the others? As much as the cute little Meerkat wants to be just like his neighboring big cats in southern Africa, he will simply never be one of them. What happens when the cleanup group members are closed as follows?

```
CloseThreadPoolCleanupGroupMembers(cg.get(), true, "Cleanup");
```

Very little is certain in multithreaded code.

If the callbacks manage to execute before they're canceled and closed, the following might be printed out:

```
work_callback: context=Cheetah  
work_callback: context=Leopard  
simple_callback: context=Meerkat  
cleanup_callback: context=Cheetah  
cleanup=Cleanup  
cleanup_callback: context=Leopard  
cleanup=Cleanup
```

A common mistake is to assume that the cleanup callback is called only for objects whose callbacks did not get an opportunity to execute. The Windows API is a bit misleading because it sometimes refers to the cleanup callback as a cancel callback, but this is not the case. The cleanup callback is simply called for every current member of the cleanup group. You could think of it as a destructor for cleanup group members, but, as with the garbage collection metaphor, this is not without risk. This metaphor holds up pretty well until you get to the `TrySubmitThreadPoolCallback` function, which once again introduces a complication. Remember that the thread pool automatically closes the underlying work object that this function creates as soon as its callback executes. That means that whether or not the cleanup callback executes for this implicit work object depends on whether or not its callback has already begun to execute by the time `CloseThreadPoolCleanupGroupMembers` is called. The cleanup callback will only be executed for this implicit work object if its work callback is still pending and you ask `CloseThreadPoolCleanupGroupMembers` to cancel any pending callbacks. This is all rather unpredictable, and I therefore don't recommend using `TrySubmitThreadPoolCallback` with a cleanup callback.

Finally, it's worth mentioning that even though `CloseThreadPoolCleanupGroupMembers` blocks, it doesn't waste any time. Any objects that are ready for cleanup will have their cleanup callbacks executed on the calling thread while it waits for other outstanding callbacks to complete. The features

provided by cleanup groups—and, in particular, the `CloseThreadPoolCleanupGroupMembers` function—are invaluable for tearing down all or parts of your application efficiently and cleanly. ■

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

THANKS to the following technical expert for reviewing this article:
Hari Pulapaka

we are Countersoft you are Control

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



GEMINI
Project Platform



**BOOK A DEMO
FREE TRIAL**
www.geminipatform.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





The Windows Azure AppFabric Service Bus: Topics

It's no secret among my colleagues that the Windows Azure Service Bus functionality didn't really get much support from me. However, with the Windows Azure AppFabric CTP June Update, Microsoft has finally added enough features to move the Service Bus from what I considered not much more than a placeholder to a truly useful technology. For my purpose here, the essential piece of messaging technology the AppFabric Service Bus now offers is *Topics*, a rich publish-and-subscribe capability. I'll focus on Topics in this article and draw, as I am so often apt to do, from my retail industry experience to look at how the technology can be used to facilitate inter-store inventory checks.

Have you ever gone to buy something and found that the last one has just been sold, or that the item you want is in some way messed up? When this happens, the sales clerk will often go into the POS system and check the inventory at nearby stores. More often than not, that check is against inventory counts that are kept in a central database or enterprise resource planning (ERP) system of some type, and the clerk usually checks by store number using her tribal knowledge of the nearby stores. Often, the data is a bit stale because it's only refreshed as part of the end-of-day processing when the transaction logs and other data are uploaded and processed by the corporate system.

The API supports correlation
and subscription IDs as
first-class citizens.

A more ideal scenario would be that a store could at any time throw a request about product availability into the ether and nearby stores would respond, indicating whether they had it. That's what I'm going to set up using the Windows Azure AppFabric Service Bus, as depicted in **Figure 1**.

Topics provide a durable mechanism that lets me push content out to up to 2,000 subscribers per topic. That subscriber limitation is unfortunate, as it potentially forces a solution architecture (like the one I'll describe) to work around it by somehow creating

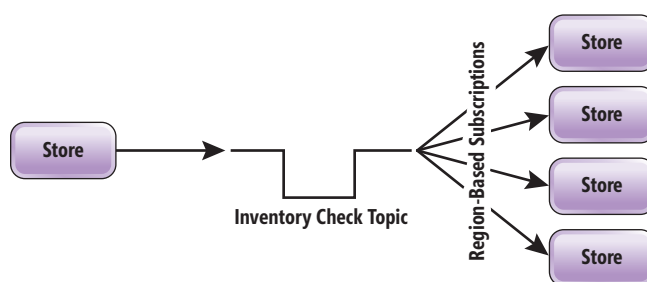


Figure 1 An Inventory-Check Message

segments in the Topic. For example, instead of a U.S. Inventory Check Topic that subscribers filter by region, I might have to create a SouthCentral U.S. Inventory Check Topic and then further filter to the specific local branch. With that caveat, I'll proceed with my single Inventory Check Topic, as I can promise I won't have more than a handful of franchise locations in my wonderland.

Getting the Message Out

You can download the Windows Azure AppFabric CTP June Update from bit.ly/p3DhAU, and find the management portal at portal.appfabriclabs.com. I'll access the management portal to retrieve a few pieces of information I need to use in my code (see **Figure 2**).

I need to grab the Service Gateway, the Default Issuer (always "owner" in the CTP) and the Default Key. Because I need these throughout my sample, I'll create them at object scope:

```
private string sbNamespace = "jofultz";  
private string issuerName = "owner";  
private string issuerKey = "25vtsCkUPtB39RdiBbUtCQg1YRP0dHHC4MvX5fvxxxx";
```

I'll use these variables in my functions to create the Topic, and then to post messages to the Topic. There's a REST interface for those who want to use it or who aren't deploying on a platform that allows use of the client library. Because I'm building a Windows client, I'll use the library. To do that, I need to add references to the Microsoft.ServiceBus, Microsoft.ServiceBus.Message and System.ServiceModel, as shown in **Figure 3**.

Once that's done, a few straightforward lines of code are all that's needed to get the Topic set up:

```
SharedSecretCredential credential =  
    TransportClientCredentialBase.CreateSharedSecretCredential(  
        issuerName, issuerKey);  
  
Uri sbUri = ServiceBusEnvironment.CreateServiceUri(  
    "sb", sbNamespace, String.Empty);  
ServiceBusNamespaceClient sbNSClient =  
    new ServiceBusNamespaceClient(sbUri, credential);  
Topic newTopic = sbNSClient.CreateTopic(topicName);
```

This article is based on the Windows Azure AppFabric CTP June Update.
All information is subject to change.



NetAdvantage[®] for jQuery

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.

Product ID	Product Name	Product Number	Standard Cost
317	CA-5965	0	
318	CA-6738	0	
319	CA-7457	0	
320	CB-2903	0	
321	CN-6137	0	
322	CR-7833	0	
323	Crown Race	CR-9981	0
324	Chain Stays	CS-2812	0
325	Decal 1	DC-8732	0
350	Fork Crown	FC-3654	0

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.

RATING

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

Movie	Rating
The Shawshank Redemption (1994)	★★★★★★★★☆
Star Wars: Episode V (1980)	★★★★★★★★☆
Raiders of the Lost Ark (1981)	★★★★★★★★☆
The Lord of the Rings (2002)	★★★★★★★★☆
Taxi Driver (1976)	★★★★★★★★☆
Paths of Glory (1957)	★★★★★★★★☆
Star Trek (2009)	★★★★★★★★☆
Life of Brian (1979)	★★★★★★★★☆
Rocky (1976)	★★★★★★★★☆
Spartacus (1960)	★★★★★★★★☆

RATING

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

IG GRID ON HTML PAGE

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



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.

IG EDITORS

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

First Name	Jack
Last Name	Black
Credit Card Number	1234 5678 9102
Expiration Date	02-01-2012
Country	da (Denmark)

Price information	
Price	kr 0.00
Quantity	5
Discount	2.00%
Total	kr 0.00
Submit	

SCAN HERE for an exclusive look at jQuery!

www.infragistics.com/jq



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

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

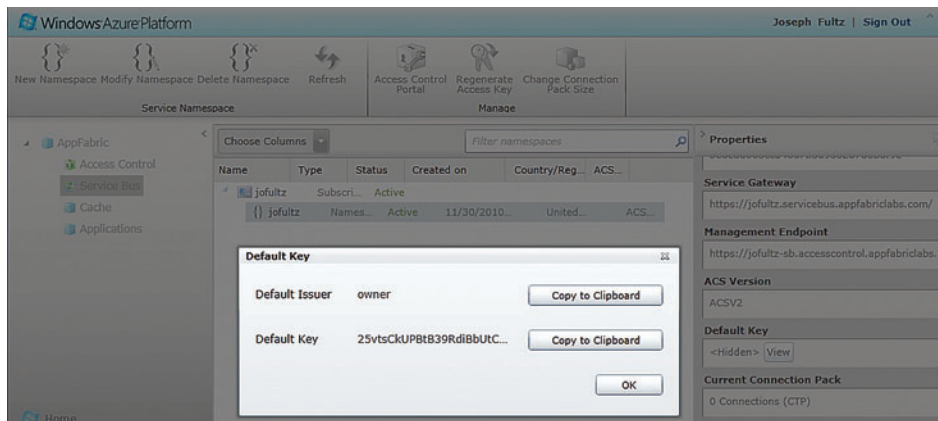


Figure 2 Getting Information from the Windows Azure AppFabric Management Portal

What will become important as I start throwing messages out there is the ability of the stores to filter the messages and get only the ones that are relevant to the receiving location based on region. The API supports correlation and subscription IDs as first-class citizens, but for the filtering I want to use my knowledge of the data and filter based on the contents of the request. Thus, I want geographically close stores to look for and respond to each other's requests for inventory based on region. At the same time, I need to make sure that the originating store doesn't pick up its own requests. Requests will include the SKU of the item in question, the region in which the inquiry is made, and the RequesterID so the originating store can filter out its own requests on the topic:

```
[Serializable]
class InventoryQueryData
{
    public string Region;
    public string ResponderID;
    public string RequesterID;
    public string Sku;
    public int Count;
}
```

Note the [Serializable] attribute that I've added to the class. You could mark the properties as DataMembers and use DataContract as well, but the point is that whatever type I intend to send to the topic needs to be serializable.

The point is that whatever type I intend to send to the topic needs to be serializable.

I've created a simple form that allows me to enter an arbitrary string for the SKU and select from a list for the posting store. The code behind my inquiry button looks similar to the code to connect and create the topic and resembles the typical constructs you find when working with messaging APIs, as shown in **Figure 4**.

There are two things to note here. First, I've added the name-value pairs I need for filtering to the BrokeredMessage.Properties collection. Second, as a matter of runtime maintenance, I've given the TimeToLive (TTL) a value of 60 seconds, which should keep the subscriptions on the topic from getting too backed up. Of

course, you'll generally want a more informed approach for picking the TTL, but I figure if the request doesn't reach any of the subscribers in that time, it's probably too long because there's a customer standing there waiting. Besides, this is just a sample.

Any message that's sent to the bus is in the form of a BrokeredMessage that has a factory method CreateMessage. This simply wraps the data into an instance of a BrokeredMessage type that contains all of the constructs needed for a fully functional messaging system.

With this I have all I need to get a message out to the subscribers of the Inventory Check Topic, so now I'll start setting up the subscription clients to fetch the messages and respond.

Once the subscription is set up it will persist even after the client closes down.

Tapping the Topic Stream and Responding

With my client in place, I'm ready to send out a steady stream of requests, but right now this will be little more than letting bits loose in the wild ether. I create a Windows Form application and reuse the XML store list and the InventoryQueryData from the first (sender) application. I'll need to create a unique subscription for each client that's listening to the topic. This is easily enough accomplished by combining a subscription name with the store number I want to listen for. My little test app allows me to select the store number from a combobox, so I just tack that value onto the base Subscription name in order to create the unique Subscription name. It's important to ensure that every client has a unique Subscription; two or more using the same subscription would create a race condition where those subscribers would compete for the message and the first one to receive and delete would win:

```
Topic invTopic = sbNSClient.GetTopic(topicName);
SubscriptionName = "InventoryQuerySubscription" + this.cboStore.Text;
SqlFilterExpression RegionFilter = new SqlFilterExpression("Region = '" +
    cboStore.SelectedValue + "' AND RequesterID <> '" + cboStore.Text + "'");
Subscription sub = invTopic.AddSubscription(SubscriptionName, RegionFilter);
```

As I add the Subscription to the Topic, I can also pass in a filter so that each store receives only the inventory check requests for its own region. Note that you can develop your own FilterExpression type from the base type, but the API includes four types that should cover most scenarios, especially if used together: CorrelationFilterExpression, MatchAllFilterExpression, MatchNoneFilterExpression and SqlFilterExpression. I used SqlFilterExpression, which allowed me to easily and instinctively write this expression to get the messages for the Texas region, for example, and exclude messages that originated from my own store:

```
"Region = '[Region]' AND RequesterID <> '[StoreID]'"
```




NetAdvantage®

for Data Visualization



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

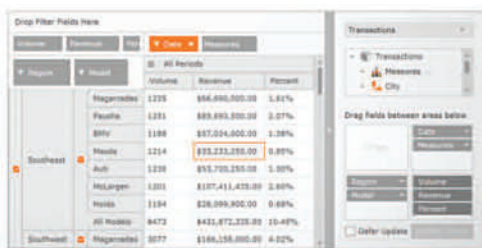


MEDIA TIMELINE
The timeline highlights how easily users can search, browse and play popular videos from YouTube.



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!



OLAP PIVOT GRID DATA VISUALIZATION

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



INFRAGISTICS MOTION FRAMEWORK™

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

SCAN HERE
for an exclusive
look at Data Visualization!
www.infragistics.com/dvis



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

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

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.

I only need to filter the requests coming through, but in some cases I could theoretically “fix up” some data on the way in using a RuleDescription to, say, combine a SqlFilterExpression with a SqlFilterAction. The former identifies the messages that I’m targeting, while the latter defines the action that should be taken. This type of functionality can be useful when the data coming through needs to be massaged into something that works for the recipient and both sides of the bus can’t or won’t be changed.

Once the subscription is set up it will persist even after the client closes down. This is perfect for this scenario; I’ll simply create a SubscriptionClient every time I start monitoring, and it will attach to the existing connection. However, not everyone will want that behavior. I can imagine situations where you’d want to remove the subscription when the client shuts down:

```
SubscriptionClient subClient = msgFactory.CreateSubscriptionClient(
    topicName, SubscriptionName);
MessageReceiver msgReceiver = subClient.CreateReceiver(ReceiveMode.ReceiveAndDelete);
msgReceiver.Open();
```

Note in the call to CreateReceiver that I’ve set ReceiveMode to ReceiveAndDelete. You could also use PeekLock for the ReceiveMode. Here, I simply want to grab the message and process it,

Figure 4 Getting the Data Values

```
// Assign data values.
InventoryQueryData data = new InventoryQueryData();
data.Sku = txtSKU.Text;
data.StoreID = cboStore.SelectedText;
data.Region = cboStore.SelectedValue.ToString();

Uri sbUri = ServiceBusEnvironment.CreateServiceUri("sb", sbNamespace, string.Empty);
SharedSecretCredential credential = TransportClientCredentialBase.CreateSharedSecretCredential(issuerName, issuerKey);

MessagingFactory msgFactory = MessagingFactory.Create(sbUri, credential);
TopicClient topicClient = msgFactory.CreateTopicClient(topicName);
MessageSender msgSender = topicClient.CreateSender();

BrokeredMessage msg = BrokeredMessage.CreateMessage(data);
// Add props to message for filtering.
msg.Properties["Region"] = data.Region;
msg.Properties["RequesterID"] = data.RequesterID;

msg.TimeToLive = TimeSpan.FromSeconds(60);

msgSender.Send(data);
```

Figure 5 Checking for Messages

```
while (MonitorOn)
{
    BrokeredMessage NewMsg;
    bool recvSuccess = msgReceiver.TryReceive(TimeSpan.FromSeconds(3), out NewMsg);

    if (recvSuccess)
    {
        this.tbMessages.Items.Add(NewMsg.MessageId);
        InventoryQueryData RequestObject =
            NewMsg.GetBody<CreateInquiry.InventoryQueryData>();
        ProcessMessage(RequestObject);
    }
    else
    {
        System.Threading.Thread.Sleep(3000);
    }
}
```

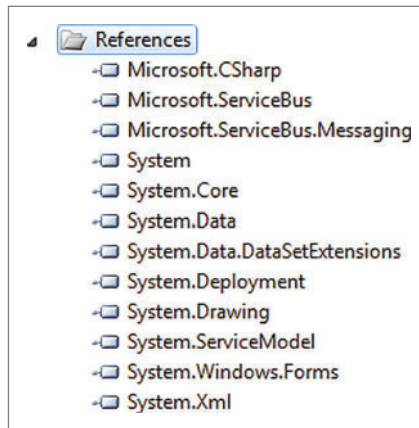


Figure 3 Adding References

and I have no need to ensure proper receipt and processing prior to deletion because if the message is lost, it isn’t a big deal. If I needed behavior that was more guaranteed and reliable, I’d likely do two things. I wouldn’t set the TTL for the Message, Topic or Subscription, and instead let the messages live indefinitely. Or, I’d give it a very high TTL so that the receiver had sufficient time to process or move to an exception queue so that only truly undeliverable messages would end up in the dead-letter queue. Also, I’d use PeekLock on the receiver to read and process the data and only remove the message upon completion of the related process. Manually creating this distributed transaction behavior

can quickly lead to other problems, such as poison queues, but we’ll address that behavior and problem set another time.

Once I’ve opened the receiver I enter a loop to check for messages. The API has a direct Receive method that will return the BrokeredMessage. However, I’ll use the TryReceive method, which will return a bool to indicate success (see Figure 5). I’ll pass a relatively short timeout to that method, which should be long enough to check and receive the message. If a message is received, I’ll work with it and immediately check for another message. If no message is received, I’ll sleep the thread for a bit and check again.

My request object is contained in the BrokeredMessage, and to retrieve it I’ll call GetBody<T>, passing the object type. The implication here is that object types need to match on both sides of the bus. You can accomplish this using proxy types or, in the case of uncertainty or plain old XML coming across, you could pass the object as a string and handle it as XML.

The implication here is that object types need to match on both sides of the bus.

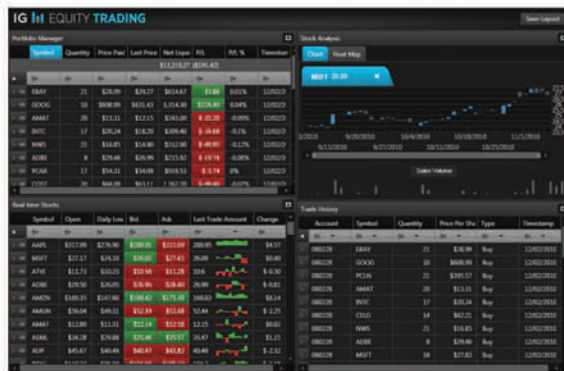
Until Next Time ...

At this point, all the work is done to create the message and get it out to all of the interested nodes, and for them to fetch the message from the Topic. I’ve demonstrated the features that allow me to not only broadcast the message, but also filter appropriately for the subscribed receiver. Next month I’ll work through the return trip to demonstrate the complementary use of queues and correlation to finish the scenario and round out your basic understanding of the new features in the Windows Azure AppFabric Service Bus. I’ll also have the complete code available for you to download. ■

JOSEPH FULTZ is a software architect at Hewlett-Packard Co., working as part of the HP.com Global IT group. 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: Jim Keane

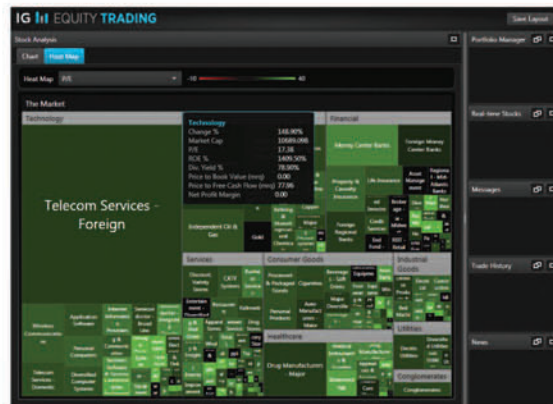
NetAdvantage[®] PERFORMANCE



**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.



TAKE IT ON THE ROAD
Our XAML controls with Windows Phone[®] 7
support means your UI is always on the go.



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

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

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

Easier Asynchronous Programming with the New Visual Studio Async CTP

Eric Lippert

Imagine what the world would be like if people worked the same way as computer programs:

```
void ServeBreakfast(Customer diner)
{
    var order = ObtainOrder(diner);
    var ingredients = ObtainIngredients(order);
    var recipe = ObtainRecipe(order);
    var meal = recipe.Prepare(ingredients);
    diner.Give(meal);
}
```

Each subroutine can, of course, be broken down further; preparing the meal might involve heating pans, cooking omelets and toasting bread. Were humans to perform these sorts of tasks like typical computer programs, we'd carefully write down everything as sequences of hierarchical tasks in a checklist and obsessively ensure that each job was complete before embarking on the next.

A subroutine-based approach seems reasonable—you can't cook the eggs before you get the order—but in fact it both wastes time

and makes the application appear unresponsive. It wastes time because you want the bread to be toasting while the eggs are frying, not after the eggs are done and getting cold. It appears unresponsive because if another customer arrives while the current order is still cooking, you want to be taking his order, not making him wait at the door until the current customer has been served her breakfast. A server slavishly following a checklist does not have any ability to respond in a timely manner to unexpected events.

Solution One: Hire More Staff by Making More Threads

Making someone's breakfast is a whimsical example, but the reality, of course, is anything but. Every time you transfer control into a long-running subroutine on the UI thread, the UI becomes completely unresponsive until the subroutine finishes. How could it be otherwise? Applications respond to UI events by running code on the UI thread, and that thread is obsessively busy doing something else. Only when every job on its list is done will it get around to dealing with the queued-up commands of the frustrated user. The usual solution to this problem is to use *concurrency* to do two or more things "at the same time." (If the two threads are on two independent processors, they might truly be running at the same time. In a world with more threads than processors to dedicate to them, the OS will simulate at-the-same-time concurrency by periodically scheduling a time slice for each thread to control a processor.)

One concurrent solution might be to create a thread pool and assign each new client a specific thread to handle its requests. In our analogy, you could hire a group of servers. When a new diner comes in, an idle server is assigned to the new diner. Each server

This article is based on a prerelease version of the Visual Studio Async CTP. All information is subject to change.

This article discusses:

- Why concurrency doesn't always lead to more responsive programs
- When `DoEvents` should and should not be used
- The problem with callbacks
- Using task-based asynchrony

Technologies discussed:

Visual Studio Async CTP, C#, Visual Basic

then independently does the work of taking the order, finding the ingredients, cooking the food and serving it.

The difficulty with this approach is that UI events typically arrive on the same thread and expect to be fully serviced on that thread. Most UI components create requests on the UI thread, and expect to be communicated with only on that thread. Dedicating a new thread to each UI-related task is unlikely to work well.

To address this problem you could have a single foreground thread listening to UI events that does nothing but “take orders” and farm them out to one or more background worker threads. In this analogy, there’s only one server who interacts with the customers, and a kitchen full of cooks who actually do the requested work. The UI thread and the worker threads are then responsible for coordinating their communications. The cooks never talk directly to the diners, but somehow the food gets served anyway.

A thread by default consumes a million bytes of virtual memory for its stack.

This certainly solves the “responding to UI events in a timely manner” problem, but it doesn’t resolve the lack of efficiency; the code running on the worker thread is still waiting synchronously for the eggs to cook fully before the bread goes in the toaster. That problem could be solved in turn by adding even *more* concurrency: You could have two cooks per order, one for the eggs and one for the toast. But that might get pretty expensive. Just how many cooks are you going to need, and what happens when they have to coordinate their work?

Concurrency of this sort introduces many well-known difficulties. First, threads are notoriously heavyweight; a thread by default consumes a million bytes of virtual memory for its stack and many other system resources. Second, UI objects are often “affinitized” to the UI thread and can’t be called from worker threads; the worker thread and the UI thread must come to some complex arrangement whereby the UI thread can send necessary information from the UI elements over to the worker, and the worker can send updates back to the UI thread, rather than to the UI elements directly. Such arrangements are difficult to code and prone to race conditions, deadlocks and other threading problems. Third, many of the pleasant fictions that we all rely upon in the single-threaded world—such as reads and writes of memory happening in a predictable and consistent sequence—are no longer reliable. This leads to the worst kinds of difficult-to-reproduce bugs.

It just seems wrong to have to use the big hammer of thread-based concurrency to build simple programs that remain responsive and run efficiently. Somehow real people manage to solve complex problems while remaining responsive to events. In the real world you don’t have to allocate one waiter per table or two cooks per order to serve dozens of customer requests that are all pending at the same time. Solving the problem with threading makes for too

many cooks. There’s got to be a better solution that doesn’t involve so much concurrency.

Solution Two: Develop Attention Deficit Disorder with DoEvents

A common non-concurrent “solution” to the problem of UI unresponsiveness during long-running operations is to liberally sprinkle the magic words `Application.DoEvents` around a program until the problem goes away. Though this is certainly a pragmatic solution, it’s not a very well-engineered one:

```
void ServeBreakfast(Customer diner)
{
    var order = ObtainOrder(diner);
    Application.DoEvents();
    var ingredients = ObtainIngredients(order);
    Application.DoEvents();
    var recipe = ObtainRecipe(order);
    Application.DoEvents();
    var meal = recipe.Prepare(ingredients);
    Application.DoEvents();
    diner.Give(meal);
}
```

Basically, using `DoEvents` means “see if anything interesting happened while I was busy doing that last thing. If something happened that I need to respond to, remember what I was doing just now, deal with the new situation, and then come back to where I left off.” It makes your program behave like it has attention deficit disorder: anything new that comes along gets attention right away. That sounds like a plausible solution to improve responsiveness—and sometimes even works—but there are a number of problems with this approach.

First, `DoEvents` works best when the delay is caused by a loop that has to execute many times, but each individual loop execution is short. By checking for pending events every few times through the loop, you can maintain responsiveness even if the whole loop takes a long time to run. However, that pattern is usually not the cause of a responsiveness problem. More often the problem is caused by one inherently long-running operation taking a lot of time, such as attempting to synchronously access a file over a high-latency network. Perhaps in our example the long-running task is in preparing the meal, and there’s no place to put the `DoEvents` that helps. Or perhaps there is a place where `DoEvents` would help, but it’s in a method you don’t have the source code for.

Second, calling `DoEvents` causes the program to attempt to fully service *all* the more recent events *before* finishing work associated with earlier events. Imagine if no one could get his meal until after every customer who came in got his meal! If more and more customers keep arriving, the first customer *might* never get his meal, resulting in starvation. In fact, it could happen that no customers get their meals. The completion of work associated with earlier events can be pushed off arbitrarily far into the future as servicing newer events continues to interrupt the work being done for earlier events.

Third, `DoEvents` poses the very real danger of unexpected reentrancy. That is, while serving one customer you check to see if there have been any recent interesting UI events and accidentally start serving the same diner again, even though he’s already being served. Most developers don’t design their code to detect this kind of reentrancy; it’s possible to end up in some very strange program states indeed when an algorithm never intended to be recursive ends up calling itself unexpectedly via `DoEvents`.

In short, DoEvents should be used only to fix a responsiveness problem in the most trivial cases; it's not a good solution for managing UI responsiveness in complex programs.

Solution Three: Turn Your Checklist Inside-out with Callbacks

The non-concurrent nature of the DoEvents technique is attractive, but clearly not quite the right solution for a complex program. A better idea is to break down the items on the checklist into a series of short tasks, each of which can be completed rapidly enough that the application can appear to be responsive to events.

That idea is nothing new; dividing a complex problem into small parts is why we have subroutines in the first place. The interesting twist is that instead of rigidly running down a checklist to determine what has already been done and what needs to be done next, and only returning control to the caller when everything is completed, *each new task is given the list of work that must come after it*. The work that must come after a particular task is finished is called the “continuation” of the task.

When a task has finished, it can look at the continuation and finish it off right there. Or it might schedule the continuation to run later. If the continuation requires information computed by the previous task, the previous task can pass that information along as an argument to the call that invokes the continuation.

With this approach, the total body of work is essentially broken up into little pieces that can each be executed rapidly. The system appears responsive because pending events can be detected and handled *between* the executions of any two of the small pieces of work. But because any activities associated with those new events can also be broken down into small parts and queued up to execute later, we don't have the “starvation” problem whereby new tasks prevent old tasks from completing. New long-running tasks are not dealt with immediately, but they are queued up for eventual processing.

The idea is great, but it's not at all clear how to implement such a solution. The essential difficulty is determining how to tell each small unit of work what its continuation is; that is, what work needs to come next.

In traditional asynchronous code, this is typically done by registering a “callback” function. Let's suppose we have an asynchronous version of “Prepare” that takes a callback function that says what to do next—namely, serve the meal:

```
void ServeBreakfast(Diner diner)
{
    var order = ObtainOrder(diner);
    var ingredients = ObtainIngredients(order);
    var recipe = ObtainRecipe(order);
    recipe.PrepareAsync(ingredients, meal =>
    {
        diner.Give(meal);
    });
}
```

Now ServeBreakfast returns *immediately* after PrepareAsync returns; whatever code called ServeBreakfast is then free to service other events that occur. PrepareAsync does no “real” work itself; rather, it quickly does whatever is necessary to ensure that the meal will be prepared in the future. Moreover, PrepareAsync also ensures that the callback method will be invoked with the prepared meal as its argument at some time after the meal

preparation task is completed. Thus, the diner will eventually be served, though she might have to wait briefly if there's an event that requires attention between the end of the preparation and the serving of the meal.

Note that *none of this necessarily involves a second thread*. Perhaps PrepareAsync causes the meal preparation work to be done on a separate thread, or perhaps it causes a series of short tasks associated with meal preparation to be queued up on the UI thread to be executed later. It really doesn't matter; all we know is that PrepareAsync somehow guarantees two things: that the meal will be prepared in a manner that doesn't block the UI thread with a high-latency operation, and that the callback will somehow be invoked after the work of preparing the requested meal is done.

Dividing a complex problem into small parts is why we have subroutines in the first place.

But suppose *any* of the methods for obtaining the order, obtaining the ingredients, obtaining the recipe or preparing the meal might be the one that's slowing down the UI. We could solve this larger problem if we had an asynchronous version of each of these methods. What would the resulting program look like? Remember, each method must be given a callback that tells it what to do when the unit of work is completed:

```
void ServeBreakfast(Diner diner)
{
    ObtainOrderAsync(diner, order =>
    {
        ObtainIngredientsAsync(order, ingredients =>
        {
            ObtainRecipeAsync(order, recipe =>
            {
                recipe.PrepareAsync(ingredients, meal =>
                {
                    diner.Give(meal);
                }));
            });
        });
    });
}
```

This might seem like an awful mess, but it's nothing compared to how bad real programs get when they're rewritten using callback-based asynchrony. Think about how you might deal with making a loop asynchronous, or how you'd deal with exceptions, try-finally blocks or other non-trivial forms of control flow. You end up essentially turning your program inside-out; the code now emphasizes how all the callbacks are wired together, and not what the logical workflow of the program should be.

Solution Four: Make the Compiler Solve the Problem with Task-Based Asynchrony

Callback-based asynchrony does keep the UI thread responsive and minimize time wasted by synchronously waiting for long-running work to complete. But the cure seems worse than the disease. The price you pay for responsiveness and performance is that you have to write code that emphasizes how the *mechanisms* of the asynchrony work while obscuring the meaning and purpose of the code.

LEADTOOLS[®] MEDICAL SDKS



FEATURES INCLUDED ARE

DICOM SDK / PACS SDK

3D VOLUME RECONSTRUCTION

**MEDICAL IMAGE VIEWER CONTROL / MEDICAL WEB
VIEWER FRAMEWORK / MEDICAL WORKSTATION**

CCOW / PRINT TO PACS

MEDICAL IMAGE PROCESSING / ANNOTATIONS

**C++ / .NET / SILVERLIGHT / WINDOWS PHONE
C DLL / WCF SERVICES / WF ACTIVITIES / WPF / ASP.NET & COM**

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

800.637.1840

The upcoming versions of C# and Visual Basic instead allow you to write code that emphasizes its meaning and purpose, while giving enough hints to the compilers to build the necessary mechanisms for you behind the scenes. The solution has two parts: one in the *type system*, and one in the *language*.

The CLR 4 release defined the type `Task<T>`—the workhorse type of the Task Parallel Library (TPL)—to represent the concept of “some work that’s going to produce a result of type `T` in the future.” The concept of “work that will complete in the future but returns no result” is represented by the non-generic `Task` type.

Precisely how the result of type `T` is going to be produced in the future is an implementation detail of a particular task; the work might be farmed out to another machine entirely, to another process on this machine, to another thread, or perhaps the work is simply to read a previously cached result that can be accessed cheaply from the current thread. TPL tasks are typically farmed out to worker threads from a thread pool in the current process, but that implementation detail is not fundamental to the `Task<T>` type; rather, a `Task<T>` can represent any high-latency operation that produces a `T`.

The language half of the solution is the new `await` keyword. A regular method call means “remember what you’re doing, run this method until it’s completely finished, and then pick up where you left off, now knowing the result of the method.” An `await` expression, in contrast, means “evaluate this expression to obtain an object representing work that will in the future produce a result. Sign up

the remainder of the current method as the callback associated with the continuation of that task. Once the task is produced and the callback is signed up, *immediately* return control to my caller.”

Our little example rewritten in the new style reads much more nicely:

```
async void ServeBreakfast(Diner diner)
{
    var order = await ObtainOrderAsync(diner);
    var ingredients = await ObtainIngredientsAsync(order);
    var recipe = await ObtainRecipeAsync(order);
    var meal = await recipe.PrepareAsync(ingredients);
    diner.Give(meal);
}
```

In this sketch, each asynchronous version returns a `Task<Order>`, `Task<List<Ingredient>>` and so on. Every time an `await` is encountered, the currently executing method signs up the rest of the method as the thing to do when the current task is complete, and then immediately returns. Somehow each task will complete itself—either by being scheduled to run as an event on the current thread, or because it used an I/O completion thread or worker thread—and will then cause its continuation to “pick up where it left off” in executing the rest of the method.

Note that the method is now marked with the new `async` keyword; this is simply an indicator to the compiler that lets it know that in the context of this method, the keyword `await` is to be treated as a point where the workflow returns control to its caller and picks up again when the associated task is finished. Note also that the examples I’ve shown in this article use C# code; Visual Basic will

have a similar feature with similar syntax. The design of these features in C# and Visual Basic was heavily influenced by F# asynchronous workflows, a feature that F# has had for some time.

Where to Learn More

This brief introduction merely motivates and then scratches the surface of the new asynchrony feature in C# and Visual Basic. For a more detailed explanation of how it works behind the scenes, and how to reason about the performance characteristics of asynchronous code, see the companion articles in this issue by my colleagues Mads Torgersen and Stephen Toub.

To get your hands on a preview release of this feature, along with samples, white papers and a community forum for questions, discussions and constructive feedback, please go to msdn.com/async. These language features and the libraries that support them are still in development; the design team would love to have as much of your feedback as possible. ■

ERIC LIPPERT is a principal developer on the C# Compiler team at Microsoft.

THANKS to the following technical experts for reviewing this article: Mads Torgersen, Stephen Toub and Lucian Wischik

Data Quality Tools for .NET



IP Location



Property



International



Dedupe



Free Form Parse



Address Verification



Email Validation



Name Parse



Phone Verification



Smart Mover

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

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

THE GREATEST FLEXGRID ON EARTH

"...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



	1	2	3	4	C
	Name				
1	<input type="checkbox"/>	Line: Computers (101 items)			
2	<input type="checkbox"/>	Color: Green (36 items)			
3	<input checked="" type="checkbox"/>	Rating: 4 (5 items)			
9	<input checked="" type="checkbox"/>	Rating: 2 (12 items)			
22	<input checked="" type="checkbox"/>	Rating: 3 (9 items)			
32	<input checked="" type="checkbox"/>	Rating: 0 (6 items)			
39	<input checked="" type="checkbox"/>	Rating: 1 (4 items)			
44	<input checked="" type="checkbox"/>	Color: Blue (28 items)			
78	<input checked="" type="checkbox"/>	Color: Red (22 items)			
106	<input checked="" type="checkbox"/>	Color: White (15 items)			
127	<input checked="" type="checkbox"/>	Line: Washers (77 items)			
229	<input checked="" type="checkbox"/>	Line: Stoves (72 items)			
261	<input checked="" type="checkbox"/>	Line: Dyers (70 items)			
289	<input checked="" type="checkbox"/>	Line: Microwaves (55 items)			
321	<input checked="" type="checkbox"/>	Line: Toasters (82 items)			

LIGHTWEIGHT!
PRINTING SUPPORT!
CELL MERGING!

SUPER FLEXIBLE!
EXCEL-LIKE EDITING!
UNBOUND MODE!

Experience The Magic @
COMPONENTONE.COM/FLEX



ComponentOne

Pause and Play with Await

Mads Torgersen

Asynchronous methods in the upcoming versions of Visual Basic and C# are a great way to get the callbacks out of your asynchronous programming. In this article, I'll take a closer look at what the new await keyword actually does, starting at the conceptual level and working my way down to the iron.

Sequential Composition

Visual Basic and C# are imperative programming languages—and proud of it! This means they excel in letting you express your programming logic as a *sequence of discrete steps*, to be undertaken one after the other. Most statement-level language constructs are control structures that give you a variety of ways to specify the order in which the discrete steps of a given body of code are to be executed:

- Conditional statements such as if and switch let you choose different subsequent actions based on the current state of the world.
- Loop statements such as for, foreach and while let you repeat the execution of a certain set of steps multiple times.
- Statements such as continue, throw and goto let you transfer control non-locally to other parts of the program.

Building up your logic using control structures results in *sequential composition*, and this is the lifeblood of imperative programming. It is indeed why there are so many control structures to choose from: You want sequential composition to be really convenient and well-structured.

Continuous Execution

In most imperative languages, including current versions of Visual Basic and C#, the execution of methods (or functions, or procedures or whatever we choose to call them) is *continuous*. What I mean by that is that once a *thread of control* has begun executing a given method, it will be continuously occupied doing so until the method execution ends. Yes, sometimes the thread will be executing statements in methods called by your body of code, but that's just part of executing the method. The thread will never switch to do anything your method didn't ask it to.

This continuity is sometimes problematic. Occasionally there's nothing a method can do to make progress—all it can do is wait for something to happen: a download, a file access, a computation happening on a different thread, a certain point in time to arrive. In such situations the thread is fully occupied doing nothing. The

This article is based on a prerelease version of the Visual Studio Async CTP. All information is subject to change.

This article discusses:

- The tension between imperative and asynchronous programming
- Using asynchronous methods to interrupt continuously executing code
- Asynchronous methods and await expressions under the hood

Technologies discussed:

Visual Studio Async CTP, C#, Visual Basic

Code download available at:

code.msdn.microsoft.com/mag201110Await

common term for that is that the thread is *blocked*; the method causing it to do so is said to be *blocking*.

Here's an example of a method that is seriously blocking:

```
static byte[] TryFetch(string url)
{
    var client = new WebClient();
    try
    {
        return client.DownloadData(url);
    }
    catch (WebException) { }
    return null;
}
```

A thread executing this method will stand still during most of the call to `client.DownloadData`, doing no actual work but just waiting.

This is bad when threads are precious—and they often are. On a typical middle tier, servicing each request in turn requires talking to a back end or other service. If each request is handled by its own thread and those threads are mostly blocked waiting for intermediate results, the sheer number of threads on the middle tier can easily become a performance bottleneck.

Probably the most precious kind of thread is a UI thread: there's only one of them. Practically all UI frameworks are single-threaded, and they require everything UI-related—events, updates, the user's UI manipulation logic—to happen on the same dedicated thread. If one of these activities (for example, an event handler choosing to download from a URL) starts to wait, the whole UI is unable to make progress because its thread is so busy doing absolutely nothing.

What we need is a way for multiple sequential activities to be able to share threads. To do that, they need to sometimes “take a break”—that is, leave holes in their execution where others can get something done on the same thread. In other words, they sometimes need to be *discontinuous*. It's particularly convenient if those sequential activities take that break while they're doing nothing anyway. To the rescue: asynchronous programming!

Asynchronous Programming

Today, because methods are always continuous, you have to split discontinuous activities (such as the before and after of a download) into multiple methods. To poke a hole in the middle of a method's execution, you have to tear it apart into its continuous bits. APIs can help by offering asynchronous (non-blocking) versions of long-running methods that initiate the operation (start the download, for example), store a passed-in callback for execution upon completion and then immediately return to the caller. But in order for the caller to provide the callback, the “after” activities need to be factored out into a separate method.

Here's how this works for the preceding `TryFetch` method:

```
static void TryFetchAsync(string url, Action<byte[], Exception>
callback)
{
    var client = new WebClient();
    client.DownloadDataCompleted += (_, args) =>
    {
        if (args.Error == null) callback(args.Result, null);
        else if (args.Error is WebException) callback(null, null);
        else callback(null, args.Error);
    };
    client.DownloadDataAsync(new Uri(url));
}
```

Here you see a couple of different ways of passing callbacks: The `DownloadDataAsync` method expects an event handler to

have been signed up to the `DownloadDataCompleted` event, so that's how you pass the “after” part of the method. `TryFetchAsync` itself also needs to deal with its callers' callbacks. Instead of setting up that whole event business yourself, you use the simpler approach of just taking a callback as a parameter. It's a good thing we can use a lambda expression for the event handler so it can just capture and use the “callback” parameter directly; if you tried to use a named method, you'd have to think of some way to get the callback delegate to the event handler. Just pause for a second and think how you'd write this code without lambdas.

But the main thing to notice here is how much the control flow changed. Instead of using the language's control structures to express the flow, you emulate them:

- The return statement is emulated by calling the callback.
- Implicit propagation of exceptions is emulated by calling the callback.
- Exception handling is emulated with a type check.

Of course, this is a very simple example. As the desired control structure gets more complex, emulating it gets even more so.

To summarize, we gained discontinuity, and thereby the ability of the executing thread to do something else while “waiting” for the download. But we lost the ease of using control structures to express the flow. We gave up our heritage as a structured imperative language.

Asynchronous Methods

When you look at the problem this way, it becomes clear how asynchronous methods in the next versions of Visual Basic and C# help: They let you express *discontinuous sequential code*.

Let's look at the asynchronous version of `TryFetch` with this new syntax:

```
static async Task<byte[]> TryFetchAsync(string url)
{
    var client = new WebClient();
    try
    {
        return await client.DownloadDataTaskAsync(url);
    }
    catch (WebException) { }
    return null;
}
```

Asynchronous methods let you take the break inline, in the middle of your code: Not only can you use your favorite control structures to express sequential composition, you can also poke holes in the execution with `await` expressions—holes where the executing thread is free to do other things.

A good way to think about this is to imagine that asynchronous methods have “pause” and “play” buttons. When the executing thread reaches an `await` expression, it hits the “pause” button and the method execution is suspended. When the task being awaited completes, it hits the “play” button, and the method execution is resumed.

Compiler Rewriting

When something complex looks simple, it usually means there's something interesting going on under the hood, and that's certainly the case with asynchronous methods. The simplicity gives you a nice abstraction that makes it so much easier to both write and read asynchronous code. Understanding what's happening underneath

is not a requirement. But if you do understand, it will surely help you become a better asynchronous programmer, and be able to more fully utilize the feature. And, if you're reading this, chances are good you're also just plain curious. So let's dive in: What do async methods—and the await expressions in them—actually do?

When the Visual Basic or C# compiler gets hold of an asynchronous method, it mangles it quite a bit during compilation: the discontinuity of the method is not directly supported by the underlying runtime and must be emulated by the compiler. So instead of you having to pull the method apart into bits, the compiler does it for you. However, it does this quite differently than you'd probably do it manually.

The compiler turns your asynchronous method into a *state machine*. The state machine keeps track of where you are in the execution and what your local state is. It can either be *running* or *suspended*. When it's running, it may reach an await, which hits the “pause” button and suspends execution. When it's suspended, something may hit the “play” button to get it back and running.

The await expression is responsible for setting things up so that the “play” button gets pushed when the awaited task completes. Before we get into that, however, let's look at the state machine itself, and what those pause and play buttons really are.

Task Builders

Asynchronous methods produce Tasks. More specifically, an asynchronous method returns an instance of one of the types Task or Task<T> from System.Threading.Tasks, and that instance is automatically generated. It doesn't have to be (and can't be) supplied by the user code. (This is a small lie: Asynchronous methods can return void, but we'll ignore that for the time being.)

From the compiler's point of view, producing Tasks is the easy part. It relies on a framework-supplied notion of a Task builder, found in System.Runtime.CompilerServices (because it's not normally meant for direct human consumption). For instance, there's a type like this:

```
public class AsyncTaskMethodBuilder<TResult>
{
    public Task<TResult> Task { get; }
    public void SetResult(TResult result);
    public void SetException(Exception exception);
}
```

The builder lets the compiler obtain a Task, and then lets it complete the Task with a result or an Exception. **Figure 1** is a sketch of what this machinery looks like for TryFetchAsync.

Watch carefully:

- First a builder is created.
- Then a `__moveNext` delegate is created. This delegate is the “play” button. We call it the resumption delegate, and it contains:
 - The original code from your async method (though we have elided it so far).
 - Return statements, which represent pushing the “pause” button.
 - Calls that complete the builder with a successful result, which correspond to the return statements of the original code.
 - A wrapping try/catch that completes the builder with any escaped exceptions.

- Now the “play” button is pushed; the resumption delegate is called. It runs until the “pause” button is hit.
- The Task is returned to the caller.

Task builders are special helper types meant only for compiler consumption. However, their behavior isn't much different from what happens when you use the TaskCompletionSource types of the Task Parallel Library (TPL) directly.

So far I've created a Task to return and a “play” button—the resumption delegate—for someone to call when it's time to resume execution. I still need to see how execution is resumed and how the await expression sets up for something to do this. Before I put it all together, though, let's take a look at how tasks are consumed.

Awaitables and Awaiters

As you've seen, Tasks can be awaited. However, Visual Basic and C# are perfectly happy to await other things as well, as long as they're *awaitable*; that is, as long as they have a certain shape that the await expression can be compiled against. In order to be awaitable, something has to have a GetAwaiter method, which in turn returns an *awaiter*. As an example, Task<TResult> has a GetAwaiter method that returns this type:

```
public struct TaskAwaiter<TResult>
{
    public bool IsCompleted { get; }
    public void OnCompleted(Action continuation);
    public TResult GetResult();
}
```

The members on the awaiter let the compiler check if the awaitable is already complete, sign up a callback to it if it isn't yet, and obtain the result (or Exception) when it is.

We can now start to see what an await should do to pause and resume around the awaitable. For instance, the await inside our TryFetchAsync example would turn into something like this:

```
__awaiter1 = client.DownloadDataTaskAsync(url).GetAwaiter();
if (!__awaiter1.IsCompleted) {
    ... // Prepare for resumption at Resume1
    __awaiter1.OnCompleted(__moveNext);
    return; // Hit the "pause" button
}
Resume1:
... __awaiter1.GetResult() ...
```

Figure 1 Building a Task

```
static Task<byte[]> TryFetchAsync(string url)
{
    var __builder = new AsyncTaskMethodBuilder<byte[]>();
    ...
    Action __moveNext = delegate
    {
        try
        {
            ...
            return;
            ...
            __builder.SetResult(...);
            ...
        }
        catch (Exception exception)
        {
            __builder.SetException(exception);
        }
    };
    __moveNext();
    return __builder.Task;
}
```


WAKE UP!

CREATE
NHIBERNATE MAPPING
EASILY!



 **devart**
WWW.DEVART.COM

Entity Developer for NHibernate

State-of-the-art modeling and
code generation tool for NHibernate



- Easy migration with powerful Import Wizard
- Model-First, Database-First, or Mixed approaches
- Powerful Update Model and Update Database Wizards
- Advanced integration into Visual Studio 2008 and 2010
- Extensive support of NHibernate mapping
- Powerful background model validation
- T4-template-based code generation with advanced output functionality
- HQL and LINQ Query Editor

FREE Express edition
30-day trial of Professional edition*

*Entity Developer is available in the following editions: for NHibernate, for Entity Framework, for LINQ to SQL, which support corresponding ORMs, and the Professional edition that allows designing models for all the three supported ORMs

Figure 2 Creating a State Machine

```
static Task<byte[]> TryFetchAsync(string url)
{
    var __builder = new AsyncTaskMethodBuilder<byte[]>();
    int __state = 0;
    Action __moveNext = null;
    TaskAwaiter<byte[]> __awaiter1;

    WebClient client = null;

    __moveNext = delegate
    {
        try
        {
            if (__state == 1) goto Resumel;
            client = new WebClient();
            try
            {
                __awaiter1 = client.DownloadDataTaskAsync(url).GetAwaiter();
                if (!__awaiter1.IsCompleted) {
                    __state = 1;
                    __awaiter1.OnCompleted(__moveNext);
                    return;
                }
            }
            Resumel:
            __builder.SetResult(__awaiter1.GetResult());
        }
        catch (WebException) { }
        __builder.SetResult(null);
    }
    catch (Exception exception)
    {
        __builder.SetException(exception);
    }
};

__moveNext();
return __builder.Task;
}
```

Again, watch what happens:

- An awaiter is obtained for the task returned from `DownloadDataTaskAsync`.
- If the awaiter is not complete, the “play” button—the resumption delegate—is passed to the awaiter as a callback.
- When the awaiter resumes execution (at `Resumel`) the result is obtained and used in the code that follows it.

Clearly the common case is that the awaitable is a `Task` or `Task<T>`. Indeed, those types—which are already present in the Microsoft .NET Framework 4—have been keenly optimized for this role. However, there are good reasons for allowing other awaitable types as well:

- Bridging to other technologies: F#, for instance, has a type `Async<T>` that roughly corresponds to `Func<Task<T>>`. Being able to await `Async<T>` directly from Visual Basic and C# helps bridge between asynchronous code written in the two languages. F# is similarly exposing bridging functionality to go the other way—consuming `Tasks` directly in asynchronous F# code.
- Implementing special semantics: The TPL itself is adding a few simple examples of this. The static `Task.Yield` utility method, for instance, returns an awaitable that will claim (via `IsCompleted`) to not be complete, but will immediately schedule the callback passed to its `OnCompleted` method, as if it had in fact completed. This lets you force scheduling and bypass the compiler’s optimization of skipping it if the result is already available. This can be used to poke holes in “live” code, and improve responsiveness of code that isn’t sitting idle. `Tasks` themselves can’t represent things

that are complete but claim not to be, so a special awaitable type is used for that.

Before I take a further look at the awaitable implementation of `Task`, let’s finish looking at the compiler’s rewriting of the asynchronous method, and flesh out the bookkeeping that tracks the state of the method’s execution.

The State Machine

In order to stitch it all together, I need to build up a state machine around the production and consumption of the `Tasks`. Essentially, all the user logic from the original method is put into the resumption delegate, but the declarations of locals are lifted out so they can survive multiple invocations. Furthermore, a state variable is introduced to track how far things have gotten, and the user logic in the resumption delegate is wrapped in a big switch that looks at the state and jumps to a corresponding label. So whenever resumption is called, it will jump right back to where it left off the last time. **Figure 2** puts the whole thing together.

Quite the mouthful! I’m sure you’re asking yourself why this code is so much more verbose than the manually “asynchronized” version shown earlier. There are a couple of good reasons, including efficiency (fewer allocations in the general case) and generality (it applies to user-defined awaitables, not just `Tasks`). However, the main reason is this: You don’t have to pull the user logic apart after all; you just augment it with some jumps and returns and such.

While the example is too simple to really justify it, rewriting a method’s logic into a semantically equivalent set of discrete methods for each of its continuous bits of logic between the `awaits` is very tricky business. The more control structures the `awaits` are nested in, the worse it gets. When not just loops with `continue` and `break` statements but `try-finally` blocks and even `goto` statements surround the `awaits`, it’s exceedingly difficult, if indeed possible, to produce a rewrite with high fidelity.

Instead of attempting that, it seems a neat trick is to just overlay the user’s original code with another layer of control structure, airlifting you in (with conditional jumps) and out (with returns) as the situation requires. Play and pause. At Microsoft, we’ve been systematically testing the equivalence of asynchronous methods to their synchronous counterparts, and we’ve confirmed that this is a very robust approach. There’s no better way to preserve synchronous semantics into the asynchronous realm than by retaining the code that describes those semantics in the first place.

The Fine Print

The description I’ve provided is slightly idealized—there are a few more tricks to the rewrite, as you may have suspected. Here are a few of the other gotchas the compiler has to deal with:

Goto Statements The rewrite in **Figure 2** doesn’t actually compile, because `goto` statements (in C# at least) can’t jump to labels buried in nested structures. That’s no problem in itself, as the compiler generates to intermediate language (IL), not source code, and isn’t bothered by nesting. But even IL doesn’t allow jumping into the middle of a `try` block, as is done in my example. Instead, what really happens is that you jump to the beginning of a `try` block, enter it normally and then switch and jump again.

Meet the EXPERTS



James Johnson
Product Manager



WE ARE REPORTING

- ✓ ActiveReports the award-winning .NET product allows Microsoft Visual Studio developers to provide royalty-free reporting solutions.
- ✓ ActiveReports now supports Silverlight and Azure.
- ✓ ActiveReports features a fast and flexible reporting engine.
- ✓ ActiveReports has an event-driven API to completely control the rendering of reports; Windows Forms, Silverlight, Web viewer controls and an end user designer control.
- ✓ ActiveReports has a wide range of export formats including Excel, TIFF, and PDF.
- ✓ ActiveReports multi-language support, extensive customization, and XCopy deployment simplifies Windows and Web reporting.

 **GrapeCity PowerTools**

www.GCPowerTools.com

GvTv.GCPowerTools.com

YOUR ONLINE SOURCE FOR DEVELOPER NEWS



GvTv.GCPowerTools.com

Smarter Components for Smarter Developers

 GrapeCity.PowerTools www.GCPowerTools.com

Finally Blocks When returning out of the resumption delegate because of an await, you don't want the finally bodies to be executed yet. They should be saved for when the *original* return statements from the user code are executed. You control that by generating a Boolean flag signaling whether the finally bodies should be executed, and augmenting them to check it.

Evaluation Order An await expression is not necessarily the first argument to a method or operator; it can occur in the middle. To preserve the order of evaluation, all the preceding arguments must be evaluated before the await, and the act of storing them and retrieving them again after the await is surprisingly involved.

On top of all this, there are a few limitations you can't get around. For instance, awaits aren't allowed inside of a catch or finally block, because we don't know of a good way to reestablish the right exception context after the await.

The Task Awaiter

The awaiter used by the compiler-generated code to implement the await expression has considerable freedom as to how it schedules the resumption delegate—that is, the rest of the asynchronous method. However, the scenario would have to be really advanced before you'd need to implement your own awaiter. Tasks themselves have quite a lot of flexibility in how they schedule because they respect a notion of scheduling context that itself is pluggable.

The scheduling context is one of those notions that would probably look a little nicer if we had designed for it from the start. As it is, it's an amalgam of a few existing concepts that we've decided not to mess up further by trying to introduce a unifying concept on top. Let's look at the idea at the conceptual level, and then I'll dive into the realization.

The philosophy underpinning the scheduling of asynchronous callbacks for awaited tasks is that you want to continue executing “where you were before,” for some value of “where.” It's this “where” that I call the scheduling context. Scheduling context is a thread-affine concept; every thread has (at most) one. When you're running on a thread, you can ask for the scheduling context it's running in, and when you have a scheduling context, you can schedule things to run in it.

So this is what an asynchronous method should do when it awaits a task:

- On suspension: Ask the thread it's running on for its scheduling context.
- On resumption: Schedule the resumption delegate back on that scheduling context.

Why is this important? Consider the UI thread. It has its own scheduling context, which schedules new work by sending it through the message queue back on the UI thread. This means that if you're running on the UI thread and await a task, when the result of the task is ready, the rest of the asynchronous method will run back on the UI thread. Thus, all the things you can do only on the UI thread (manipulating the UI) you can still do after the await; you won't experience a weird “thread hop” in the middle of your code.

Other scheduling contexts are multithreaded; specifically, the standard thread pool is represented by a single scheduling context. When new work is scheduled to it, it may go on any of the pool's

threads. Thus, an asynchronous method that starts out running on the thread pool will continue to do so, though it may “hop around” among different specific threads.

In practice, there's no single concept corresponding to the scheduling context. Roughly speaking, a thread's SynchronizationContext acts as its scheduling context. So if a thread has one of those (an existing concept that can be user-implemented), it will be used. If it doesn't, then the thread's TaskScheduler (a similar concept introduced by the TPL) is used. If it doesn't have one of those either, the default TaskScheduler is used; that one schedules resumptions to the standard thread pool.

Of course, all this scheduling business has a performance cost. Usually, in user scenarios, it's negligible and well worth it: Having your UI code chopped up into manageable bits of actual live work and pumped in through the message pump as waited-for results become available is normally just what the doctor ordered.

Sometimes, though—especially in library code—things can get too fine-grained. Consider:

```
async Task<int> GetAreaAsync()
{
    return await GetXAsync() * await GetYAsync();
}
```

This schedules back to the scheduling context twice—after each await—just to perform a multiplication on the “right” thread. But who cares what thread you're multiplying on? That's probably wasteful (if used often), and there are tricks to avoid it: You can essentially wrap the awaited Task in a non-Task awaitable that knows how to turn off the schedule-back behavior and just run the resumption on whichever thread completes the task, avoiding the context switch and the scheduling delay:

```
async Task<int> GetAreaAsync()
{
    return await GetXAsync().ConfigureAwait(continueOnCapturedContext: false)
        * await GetYAsync().ConfigureAwait(continueOnCapturedContext: false);
}
```

Less pretty, to be sure, but a neat trick to use in library code that ends up being a bottleneck for scheduling.

Go Forth and Async'ify

Now you should have a working understanding of the underpinnings of asynchronous methods. Probably the most useful points to take away are:

- The compiler preserves the meaning of your control structures by actually preserving your control structures.
- Asynchronous methods don't schedule new threads—they let you multiplex on existing ones.
- When tasks get awaited, they put you back “where you were” for a reasonable definition of what that means.

If you're like me, you've already been alternating between reading this article and typing in some code. You've multiplexed multiple flows of control—reading and coding—on the same thread: you. That's just what asynchronous methods let you do. ■

MADS TORGENSEN is a principal program manager on the C# and Visual Basic Language team at Microsoft.

THANKS to the following technical expert for reviewing this article:
Stephen Toub

Async Performance: Understanding the Costs of Async and Await

Stephen Toub

Asynchronous programming has long been the realm of only the most skilled and masochistic of developers—those with the time, inclination and mental capacity to reason about callback after callback of non-linear control flow. With the Microsoft .NET Framework 4.5, C# and Visual Basic deliver asynchronicity for the rest of us, such that mere mortals can write asynchronous methods almost as easily as writing synchronous methods. No more callbacks. No more explicit marshaling of code from one synchronization context to another. No more worrying about the flowing of results or exceptions. No more tricks that contort existing language features to ease async development. In short, no more hassle.

Of course, while it's now easy to get started writing asynchronous methods (see the articles by Eric Lippert and Mads Torgersen in this issue of *MSDN Magazine*), doing it really well still requires an

understanding of what's happening under the covers. Any time a language or framework raises the level of abstraction at which a developer can program, it invariably also encapsulates hidden performance costs. In many cases, such costs are negligible and can and should be ignored by the vast number of developers implementing the vast number of scenarios. However, it still behooves more advanced developers to really understand what costs exist so they can take any necessary steps to avoid those costs if they do eventually become visible. Such is the case with the asynchronous methods feature in C# and Visual Basic.

In this article, I'll explore the ins and outs of asynchronous methods, providing you with a solid understanding of how asynchronous methods are implemented under the covers and discussing some of the more nuanced costs involved. Note that this information isn't meant to encourage you to contort readable code into something that can't be maintained, all in the name of micro-optimization and performance. It's simply to give you information that may help you diagnose any problems you may run across, as well as supply a set of tools to help you overcome such potential issues. Note also that this article is based on a preview release of the .NET Framework 4.5, and it's likely that specific implementation details will change prior to the final release.

Getting the Right Mental Model

For decades, developers have used high-level languages like C#, Visual Basic, F# and C++ to develop efficient applications. This experience has informed those developers about the relevant

This article is based on a preview release of the Microsoft .NET Framework 4.5. It's likely that specific implementation details will change prior to the final release.

This article discusses:

- Understanding the performance implications of asynchronous methods
- Optimizing object allocations and garbage collections
- The impact of context on asynchronous method performance

Technologies discussed:

Microsoft .NET Framework 4.5, C#, Visual Basic

costs of various operations, and that knowledge has informed best development practices. For example, for most use cases, calling a synchronous method is relatively cheap, even more so when the compiler is able to inline the callee into the call site. Thus, developers learn to refactor code into small, maintainable methods, in general without needing to think about any negative ramifications from the increased method invocation count. These developers have a mental model for what it means to call a method.

With the introduction of asynchronous methods, a new mental model is needed. While the C# and Visual Basic languages and compilers are able to provide the illusion of an asynchronous method being just like its synchronous counterpart, under the covers it's no such thing. The compiler ends up generating a lot of code on behalf of the developer, code akin to the quantities of boilerplate code that developers implementing asynchronicity in days of yore would've had to have written and maintained by hand. Further still, the compiler-generated code calls into library code in the .NET Framework, again increasing the work done on behalf of the developer. To get the right mental model, and then to use that mental model to make appropriate development decisions, it's important to understand what the compiler is generating on your behalf.

Think Chunky, Not Chatty

When working with synchronous code, methods with empty bodies are practically free. This is not the case for asynchronous methods. Consider the following asynchronous method, which has a single statement in its body (and which due to lack of awaits will end up running synchronously):

```
public static async Task SimpleBodyAsync() {  
    Console.WriteLine("Hello, Async World!");  
}
```

An intermediate language (IL) decompiler will reveal the true nature of this function once compiled, with output similar to what's shown in **Figure 1**. What was a simple one-liner has been expanded into two methods, one of which exists on a helper state machine class. First, there's a stub method that has the same basic signature as that written by the developer (the method is named the same, it has the same visibility, it accepts the same parameters and it retains its return type), but that stub doesn't contain any of the code written by the developer. Rather, it contains setup boilerplate. The setup code initializes the state machine used to represent the asynchronous method and then kicks it off using a call to the secondary `MoveNext` method on the state machine. This state machine type holds state for the asynchronous method, allowing that state to be persisted across asynchronous await points, if necessary. It also contains the body of the method as written by the user, but contorted in a way that allows for results and exceptions to be lifted into the returned `Task`; for the current position in the method to be maintained so that execution may resume at that location after an await; and so on.

When thinking through what asynchronous methods cost to invoke, keep this boilerplate in mind. The try/catch block in the `MoveNext` method will likely prevent it from getting inlined by the just-in-time (JIT) compiler, so at the very least we'll now have the cost of a method invocation where in the synchronous case we likely would not (with such a small method body). We have multiple calls into Framework routines (like `SetResult`). And we have

Figure 1 Asynchronous Method Boilerplate

```
[DebuggerStepThrough]  
public static Task SimpleBodyAsync() {  
    <SimpleBodyAsync>d__0 d__ = new <SimpleBodyAsync>d__0();  
    d__.<t__builder = AsyncTaskMethodBuilder.Create();  
    d__.MoveNext();  
    return d__.<t__builder.Task;  
}  
  
[CompilerGenerated]  
[StructLayout(LayoutKind.Sequential)]  
private struct <SimpleBodyAsync>d__0 : <t__IStateMachine {  
    private int <>l__state;  
    public AsyncTaskMethodBuilder <t__builder;  
    public Action <t__MoveNextDelegate;  
  
    public void MoveNext() {  
        try {  
            if (this.<>l__state == -1) return;  
            Console.WriteLine("Hello, Async World!");  
        }  
        catch (Exception e) {  
            this.<>l__state = -1;  
            this.<t__builder.SetException(e);  
            return;  
        }  
  
        this.<>l__state = -1;  
        this.<t__builder.SetResult();  
    }  
  
    ...  
}
```

multiple writes to fields on the state machine type. Of course, we need to weigh all of this against the cost of the `Console.WriteLine`, which will likely dominate all of the other costs involved (it takes locks, it does I/O and so forth). Further, notice that there are optimizations the infrastructure does for you. For example, the state machine type is a struct. That struct will only be boxed to the heap if this method ever needs to suspend its execution because it's awaiting an instance that's not yet completed, and in this simple method, it never will complete. As such, the boilerplate of this asynchronous method won't incur any allocations. The compiler and runtime work hard together to minimize the number of allocations involved in the infrastructure.

The .NET Framework
attempts to generate efficient
asynchronous implementations
for asynchronous methods.

Know When Not to Use Async

The .NET Framework attempts to generate efficient asynchronous implementations for asynchronous methods, applying multiple optimizations. However, developers often have domain knowledge than can yield optimizations that would be risky and unwise for the compiler and runtime to apply automatically, given the generality they target. With this in mind, it can actually benefit a developer to avoid using async methods in a certain, small set of use cases,

Figure 2 Optimizing Task Allocations

```
private Task<int> m_lastTask;

public override Task<int> ReadAsync(
    byte [] buffer, int offset, int count,
    CancellationToken cancellationToken)
{
    if (cancellationToken.IsCancellationRequested) {
        var tcs = new TaskCompletionSource<int>();
        tcs.SetCanceled();
        return tcs.Task;
    }

    try {
        int numRead = this.Read(buffer, offset, count);
        return m_lastTask != null && numRead == m_lastTask.Result ?
            m_lastTask : (m_lastTask = Task.FromResult(numRead));
    }
    catch (Exception e) {
        var tcs = new TaskCompletionSource<int>();
        tcs.SetException(e);
        return tcs.Task;
    }
}
```

particularly for library methods that will be accessed in a more fine-grained manner. Typically, this is the case when it's known that the method may actually be able to complete synchronously because the data it's relying on is already available.

When designing asynchronous methods, the Framework developers spent a lot of time optimizing away object allocations. This is because allocations represent one of the largest performance costs possible in the asynchronous method infrastructure. The act of allocating an object is typically quite cheap. Allocating objects is akin to filling your shopping cart with merchandise, in that it doesn't cost you much effort to put items into your cart; it's when you actually check out that you need to pull out your wallet and invest significant resources. While allocations are usually cheap, the resulting garbage collection can be a showstopper when it comes to the application's performance. The act of garbage collection involves scanning through some portion of objects currently allocated and finding those that are no longer referenced. The more objects allocated, the longer it takes to perform this marking. Further, the larger the allocated objects and the more of them that are allocated, the more frequently garbage collection needs to occur. In this manner, then, allocations have a global effect on the system: the more garbage generated by asynchronous methods, the slower the overall program will run, even if micro benchmarks of the asynchronous methods themselves don't reveal significant costs.

For asynchronous methods that actually yield execution (due to awaiting an object that's not yet completed), the asynchronous method infrastructure needs to allocate a Task object to return from the method, as that Task serves as a unique reference for this particular invocation. However, many asynchronous method invocations can complete without ever yielding. In such cases, the asynchronous method infrastructure may return a cached, already completed Task, one that it can use over and over to avoid allocating unnecessary Tasks. It's only able to do this in limited circumstances, however, such as when the asynchronous method is a non-generic Task, a Task<Boolean>, or when it's a Task<TResult> where TResult is a reference type and the result of the asynchronous method is null. While this set may expand in

the future, you can often do better if you have domain knowledge of the operation being implemented.

Consider implementing a type like MemoryStream. MemoryStream derives from Stream, and thus can override Stream's new .NET 4.5 ReadAsync, WriteAsync and FlushAsync methods to provide optimized implementations for the nature of MemoryStream. Because the operation of reading is simply going against an in-memory buffer and is therefore just a memory copy, better performance results if ReadAsync runs synchronously. Implementing this with an asynchronous method would look something like the following:

```
public override async Task<int> ReadAsync(
    byte [] buffer, int offset, int count,
    CancellationToken cancellationToken)
{
    cancellationToken.ThrowIfCancellationRequested();
    return this.Read(buffer, offset, count);
}
```

Easy enough. And because Read is a synchronous call, and because there are no awaits in this method that will yield control, all invocations of ReadAsync will actually complete synchronously. Now, let's consider a standard usage pattern of streams, such as a copy operation:

```
byte [] buffer = new byte[0x1000];
int numRead;
while((numRead = await source.ReadAsync(buffer, 0, buffer.Length)) > 0) {
    await source.WriteAsync(buffer, 0, numRead);
}
```

Notice here that ReadAsync on the source stream for this particular series of calls is always invoked with the same count parameter (the buffer's length), and thus it's very likely that the return value (the number of bytes read) will also be repeating. Except in some rare circumstances, it's very unlikely that the asynchronous method implementation of ReadAsync will be able to use a cached Task for its return value, but you can.

The Framework developers
spent a lot of time optimizing
away object allocations.

Consider rewriting this method as shown in **Figure 2**. By taking advantage of the specific aspects of this method and its common usage scenarios, we've now been able to optimize allocations away on the common path in a way we couldn't expect the underlying infrastructure to do. With this, every time a call to ReadAsync retrieves the same number of bytes as the previous call to ReadAsync, we're able to completely avoid any allocation overhead from the ReadAsync method by returning the same Task we returned on the previous invocation. And for a low-level operation like this that we expect to be very fast and to be invoked repeatedly, such an optimization can make a noticeable difference, especially in the number of garbage collections that occur.

A related optimization to avoid the task allocation may be done when the scenario dictates caching. Consider a method whose purpose it is to download the contents of a particular Web page and then cache its successfully downloaded contents for future accesses.

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**
 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.

Such functionality might be written using an asynchronous method as follows (using the new System.Net.Http.dll library in .NET 4.5):

```
private static ConcurrentDictionary<string,string> s_urlToContents;

public static async Task<string> GetContentsAsync(string url)
{
    string contents;
    if (!s_urlToContents.TryGetValue(url, out contents))
    {
        var response = await new HttpClient().GetAsync(url);
        contents = response.EnsureSuccessStatusCode().Content.ReadAsStringAsync();
        s_urlToContents.TryAdd(url, contents);
    }
    return contents;
}
```

This is a straightforward implementation. And for calls to `GetContentsAsync` that can't be satisfied from the cache, the overhead of constructing a new `Task<string>` to represent this download will be negligible when compared to the network-related costs. However, for cases where the contents may be satisfied from the cache, it could represent a non-negligible cost, an object allocation simply to wrap and hand back already available data.

To avoid that cost (if doing so is required to meet your performance goals), you could rewrite this method as shown in **Figure 3**. We now have two methods: a synchronous public method, and an asynchronous private method to which the public method delegates. The dictionary is now caching the generated tasks rather than their contents, so future attempts to download a page that's already been successfully downloaded can be satisfied with a simple dictionary access to return an already existing task. Internally, we also take advantage of the `ContinueWith` methods on `Task` that allow us to store the task into the dictionary once the `Task` has completed—but only if the download succeeded. Of course, this code is more complicated and requires more thought to write and maintain, so as with any performance optimizations, avoid spending time making them until performance testing proves that the complications make an impactful and necessary difference. Whether such optimizations make a difference really depends on usage scenarios. You'll want to come up with a suite of tests that represent common usage patterns, and use analysis of those tests to determine whether these complications improve your code's performance in a meaningful way.

Another task-related optimization to consider is whether you even need the returned `Task` from an asynchronous method. C# and

Visual Basic both support the creation of asynchronous methods that return `void`, in which case no `Task` is allocated for the method, ever. Asynchronous methods exposed publicly from libraries should always be written to return a `Task` or `Task<TResult>`, because you as a library developer don't know whether the consumer desires to wait on the completion of that method. However, for certain internal usage scenarios, void-returning asynchronous methods can have their place. The primary reason void-returning asynchronous methods exist is to support existing event-driven environments, like ASP.NET and Windows Presentation Foundation (WPF). They make it easy to implement button handlers, page-load events and the like through the use of `async` and `await`. If you do consider using an async void method, be very careful around exception handling: exceptions that escape an async void method bubble out into whatever `SynchronizationContext` was current at the time the async void method was invoked.

You'll want to come up with
a suite of tests that represent
common usage patterns.

Care About Context

There are many kinds of “context” in the .NET Framework: `LogicalCallContext`, `SynchronizationContext`, `HostExecutionContext`, `SecurityContext`, `ExecutionContext` and more (from the sheer number you might expect that the developers of the Framework are monetarily incentivized to introduce new contexts, but I assure you we're not). Some of these contexts are very relevant to asynchronous methods, not only in functionality, but also in their impact on asynchronous method performance.

SynchronizationContext `SynchronizationContext` plays a big role in asynchronous methods. A “synchronization context” is simply an abstraction over the ability to marshal delegate invocation in a manner specific to a given library or framework. For example, WPF provides a `DispatcherSynchronizationContext` to represent the UI thread for a `Dispatcher`: posting a delegate to this synchronization context causes that delegate to be queued for execution by the `Dispatcher` on its thread. ASP.NET provides an `AspNetSynchronizationContext`, which is used to ensure that asynchronous operations that occur as part of the processing of an ASP.NET request are executed serially and are associated with the right `HttpContext` state. And so on. All told, there are around 10 concrete implementations of `SynchronizationContext` within the .NET Framework, some public, some internal.

When awaiting `Tasks` and other awaitable types provided by the .NET Framework, the “awaiters” for those types (like `TaskAwaiter`) capture the current `SynchronizationContext` at the time the `await` is issued. Upon completion of the awaitable, if there was a current `SynchronizationContext` that got captured, the continuation representing the remainder of the asynchronous method is posted to that `SynchronizationContext`. With that, developers writing

Figure 3 Manually Caching Tasks

```
private static ConcurrentDictionary<string,Task<string>> s_urlToContents;

public static Task<string> GetContentsAsync(string url) {
    Task<string> contents;
    if (!s_urlToContents.TryGetValue(url, out contents)) {
        contents = GetContentsAsync(url);
        contents.ContinueWith(delegate {
            s_urlToContents.TryAdd(url, contents);
        }, CancellationToken.None,
            TaskContinuationOptions.OnlyOnRanToCompletion |
            TaskContinuationOptions.ExecuteSynchronously,
            TaskScheduler.Default);
    }
    return contents;
}

private static async Task<string> GetContentsAsync(string url) {
    var response = await new HttpClient().GetAsync(url);
    return response.EnsureSuccessStatusCode().Content.ReadAsStringAsync();
}
```

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.

Figure 4 Local Lifting

```
[StructLayout(LayoutKind.Sequential), CompilerGenerated]
private struct <FooAsync>d__0 : <T_IStateMachine {
    private int <I__state;
    public AsyncTaskMethodBuilder <T__builder;
    public Action <T__MoveNextDelegate;

    public DateTimeOffset <dto>5__1;
    public DateTime <dt>5__2;
    private object <T__stack;
    private object <T__awaiter;

    public void MoveNext();
    [DebuggerHidden]
    public void <T__SetMoveNextDelegate(Action param0);
}
```

an asynchronous method called from a UI thread don't need to manually marshal invocations back to the UI thread in order to modify UI controls: such marshaling is handled automatically by the Framework infrastructure.

Unfortunately, such marshaling also involves cost. For application developers using await to implement their control flow, this automatic marshaling is almost always the right solution. Libraries, however, are often a different story. Application developers typically need such marshaling because their code cares about the context under which it's running, such as being able to access UI controls, or being able to access the HttpContext for the right ASP.NET request. Most libraries, however, do not suffer this constraint. As a result, this automatic marshaling is frequently an entirely unnecessary cost. Consider again the code shown earlier to copy data from one stream to another:

```
byte [] buffer = new byte[0x1000];
int numRead;
while((numRead = await source.ReadAsync(buffer, 0, buffer.Length)) > 0) {
    await source.WriteAsync(buffer, 0, numRead);
}
```

If this copy operation is invoked from a UI thread, every awaited read and write operation will force the completion back to the UI thread. For a megabyte of source data and Streams that complete reads and writes asynchronously (which is most of them), that means upward of 500 hops from background threads to the UI thread. To address this, the Task and Task<TResult> types provide a ConfigureAwait method. ConfigureAwait accepts a Boolean continueOnCapturedContext parameter that controls this marshaling behavior. If the default of true is used, the await will automatically complete back on the captured SynchronizationContext. If false is used, however, the SynchronizationContext will be ignored and the Framework will attempt to continue the execution wherever the previous asynchronous operation completed. Incorporating this into the stream-copying code results in the following more efficient version:

```
byte [] buffer = new byte[0x1000];
int numRead;
while((numRead = await
    source.ReadAsync(buffer, 0, buffer.Length).ConfigureAwait(false)) > 0) {
    await source.WriteAsync(buffer, 0, numRead).ConfigureAwait(false);
}
```

For library developers, this performance impact alone is sufficient to warrant always using ConfigureAwait, unless it's the rare circumstance where the library has domain knowledge of its environment and does need to execute the body of the method with access to the correct context.

There's another reason, beyond performance, to use ConfigureAwait in library code. Suppose the preceding code, without ConfigureAwait, was in a method called CopyStreamToStreamAsync, which was invoked from a WPF UI thread, like so:

```
private void button1_Click(object sender, EventArgs args) {
    Stream src = ..., dst = ...;
    Task t = CopyStreamToStreamAsync(src, dst);
    t.Wait(); // deadlock!
}
```

Here, the developer should have written button1_Click as an async method and then await-ed the Task instead of using its synchronous Wait method. The Wait method has its important uses, but it's almost always wrong to use it for waiting in a UI thread like this. The Wait method won't return until the Task has completed. In the case of CopyStreamToStreamAsync, the contained awaits try to Post back to the captured SynchronizationContext, and the method can't complete until those Posts complete (because the Posts are used to process the remainder of the method). But those Posts won't complete, because the UI thread that would process them is blocked in the call to Wait. This is a circular dependency, resulting in a deadlock. If CopyStreamToStreamAsync had instead been written using ConfigureAwait(false), there would be no circular dependency and no deadlock.

The Wait method has its important uses, but it's almost always wrong to use it for waiting in a UI thread.

ExecutionContext ExecutionContext is an integral part of the .NET Framework, yet most developers are blissfully unaware of its existence. ExecutionContext is the granddaddy of contexts, encapsulating multiple other contexts like SecurityContext and LogicalCallContext, and representing everything that should be automatically flowed across asynchronous points in code. Any time you've used ThreadPool.QueueUserWorkItem, Task.Run, Delegate.BeginInvoke, Stream.BeginRead, WebClient.DownloadStringAsync or any other asynchronous operation in the Framework, under the covers ExecutionContext was captured if possible (via ExecutionContext.Capture), and that captured context was then used to process the provided delegate (via ExecutionContext.Run). For example, if the code invoking ThreadPool.QueueUserWorkItem was impersonating a Windows identity at the time, that same Windows identity would be impersonated in order to run the supplied WaitCallback delegate. And if the code invoking Task.Run had first stored data into the LogicalCallContext, that same data would be accessible through the LogicalCallContext within the supplied Action delegate. ExecutionContext is also flowed across awaits on tasks.

There are multiple optimizations in place in the Framework to avoid capturing and running under a captured ExecutionContext when doing so is unnecessary, as doing so can be quite expensive.

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

DevExpress™

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

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

Figure 5 Applying Multiple Optimizations

```
public static Task<int> SumAsync(
    Task<int> a, Task<int> b, Task<int> c)
{
    return (a.Status == TaskStatus.RanToCompletion &&
        b.Status == TaskStatus.RanToCompletion &&
        c.Status == TaskStatus.RanToCompletion) ?
        Task.FromResult(Sum(a.Result, b.Result, c.Result)) :
        SumAsyncInternal(a, b, c);
}

private static async Task<int> SumAsyncInternal(
    Task<int> a, Task<int> b, Task<int> c)
{
    await Task.WhenAll((Task[]) { a, b, c }).ConfigureAwait(false);
    return Sum(a.Result, b.Result, c.Result);
}
```

However, actions like impersonating a Windows identity or storing data into LogicalCallContext will thwart these optimizations. Avoiding operations that manipulate ExecutionContext, such as WindowsIdentity.Impersonate and CallContext.LogicalSetData, results in better performance when using asynchronous methods, and when using asynchrony in general.

Lift Your Way out of Garbage Collection

Asynchronous methods provide a nice illusion when it comes to local variables. In a synchronous method, local variables in C# and Visual Basic are stack-based, such that no heap allocations are necessary to store those locals. However, in asynchronous methods, the stack for the method goes away when the asynchronous method is suspending at an await point. For data to be available to the method after an await resumes, that data must be stored somewhere. Thus, the C# and Visual Basic compilers “lift” locals into a state machine struct, which is then boxed to the heap at the first await that suspends so that locals may survive across await points.

The bigger the objects being allocated, the more often garbage collection will need to run.

Earlier in this article, I discussed how the cost and frequency of garbage collection is influenced by the number of objects allocated, while the frequency of garbage collection is also influenced by the size of objects allocated. The bigger the objects being allocated, the more often garbage collection will need to run. Thus, in an asynchronous method, the more locals that need to be lifted to the heap, the more often garbage collections will occur.

As of the time of this writing, the C# and Visual Basic compilers sometimes lift more than is truly necessary. For example, consider the following code snippet:

```
public static async Task FooAsync() {
    var dto = DateTimeOffset.Now;
    var dt = dto.DateTime;
    await Task.Yield();
    Console.WriteLine(dt);
}
```

The dto variable isn't read at all after the await point, and thus the value written to it before the await doesn't need to survive across the await. However, the state machine type generated by the compiler to store locals still contains the dto reference, as shown in Figure 4.

This slightly bloats the size of that heap object beyond what's truly necessary. If you find that garbage collections are occurring more frequently than you expect, take a look at whether you really need all of the temporary variables you've coded into your asynchronous method. This example could be rewritten as follows to avoid the extra field on the state machine class:

```
public static async Task FooAsync() {
    var dt = DateTimeOffset.Now.DateTime;
    await Task.Yield();
    Console.WriteLine(dt);
}
```

Moreover, the .NET garbage collector (GC) is a generational collector, meaning that it partitions the set of objects into groups, known as generations: at a high-level, new objects are allocated in generation 0, and then all objects that survive a collection are promoted up a generation (the .NET GC currently uses generations 0, 1 and 2). This enables faster collections by allowing the GC to frequently collect only from a subset of the known object space. It's based on the philosophy that objects newly allocated will also go away quickly, while objects that have been around for a long time will continue to be around for a long time. What this means is that if an object survives generation 0, it will likely end up being around for a while, continuing to put pressure on the system for that additional time. And that means we really want to ensure that objects are made available to garbage collection as soon as they're no longer needed.

With the aforementioned lifting, locals get promoted to fields of a class that stays rooted for the duration of the asynchronous method's execution (as long as the awaited object properly maintains a reference to the delegate to invoke upon completion of the awaited operation). In synchronous methods, the JIT compiler is able to keep track of when locals will never again be accessed, and at such points can help the GC to ignore those variables as roots, thus making the referenced objects available for collection if they're not referenced anywhere else. However, in asynchronous methods, these locals remain referenced, which means the objects they reference may survive much longer than if these had been real locals. If you find that objects are remaining alive well past their use, consider nulling out the locals referencing those objects when you're done with them. Again, this should be done only if you find that it's actually the cause of a performance problem, as it otherwise complicates the code unnecessarily. Furthermore, the C# and Visual Basic compilers could be updated by final release or otherwise in the future to handle more of these scenarios on the developer's behalf, so any such code written today is likely to become obsolete in the future.

Avoid Complexity

The C# and Visual Basic compilers are fairly impressive in terms of where you're allowed to use awaits: almost anywhere. Await expressions may be used as part of larger expressions, allowing you to await Task<TResult> instances in places you might have

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



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

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

any other value-returning expression. For example, consider the following code, which returns the sum of three tasks' results:

```
public static async Task<int> SumAsync(
    Task<int> a, Task<int> b, Task<int> c)
{
    return Sum(await a, await b, await c);
}

private static int Sum(int a, int b, int c)
{
    return a + b + c;
}
```

The C# compiler allows you to use the expression “await b” as an argument to the Sum function. However, there are multiple awaits here whose results are passed as parameters to Sum, and due to order of evaluation rules and how async is implemented in the compiler, this particular example requires the compiler to “spill” the temporary results of the first two awaits. As you saw previously, locals are preserved across await points by having them lifted into fields on the state machine class. However, for cases like this one, where the values are on the CLR evaluation stack, those values aren't lifted into the state machine but are instead spilled to a single temporary object and then referenced by the state machine. When you complete the await on the first task and go to await the second one, the compiler generates code that boxes the first result and stores the boxed object into a single `<T>__stack` field on the state machine. When you complete the await on the second task and go to await the third one, the compiler generates code that creates a `Tuple<int,int>` from the first two values, storing that tuple into the same `<T>__stack` field. This all means that, depending on how you write your code, you could end up with very different allocation patterns. Consider instead writing SumAsync as follows:

```
public static async Task<int> SumAsync(
    Task<int> a, Task<int> b, Task<int> c)
{
    int ra = await a;
    int rb = await b;
    int rc = await c;
    return Sum(ra, rb, rc);
}
```

The fewer awaits you need to process, the better.

With this change, the compiler will now emit three more fields onto the state machine class to store ra, rb and rc, and no spilling will occur. Thus, you have a trade-off: a larger state machine class with fewer allocations, or a smaller state machine class with more allocations. The total amount of memory allocated will be larger in the spilling case, as each object allocated has its own memory overhead, but in the end performance testing could reveal that's still better. In general, as mentioned previously, you shouldn't think through these kinds of micro-optimizations unless you find that the allocations are actually the cause of grief, but regardless, it's helpful to know where these allocations are coming from.

Of course, there's arguably a much larger cost in the preceding examples that you should be aware of and proactively consider. The code isn't able to invoke Sum until all three awaits have completed,

and no work is done in between the awaits. Each of these awaits that yields requires a fair amount of work, so the fewer awaits you need to process, the better. It would behoove you, then, to combine all three of these awaits into just one by waiting on all of the tasks at once with Task.WhenAll:

```
public static async Task<int> SumAsync(
    Task<int> a, Task<int> b, Task<int> c)
{
    int [] results = await Task.WhenAll(a, b, c);
    return Sum(results[0], results[1], results[2]);
}
```

The Task.WhenAll method here returns a `Task<TResult[]>` that won't complete until all of the supplied tasks have completed, and it does so much more efficiently than just waiting on each individual task. It also gathers up the result from each task and stores it into an array. If you want to avoid that array, you can do that by forcing binding to the non-generic WhenAll method that works with Task instead of Task<TResult>. For ultimate performance, you could also take a hybrid approach, where you first check to see if all of the tasks have completed successfully, and if they have, get their results individually—but if they haven't, then await a WhenAll of those that haven't. That will avoid any allocations involved in the call to WhenAll when it's unnecessary, such as allocating the params array to be passed into the method. And, as previously mentioned, we'd want this library function to also suppress context marshaling. Such a solution is shown in Figure 5.

Asynchronicity and Performance

Asynchronous methods are a powerful productivity tool, enabling you to more easily write scalable and responsive libraries and applications. It's important to keep in mind, though, that asynchronicity is not a performance optimization for an individual operation. Taking a synchronous operation and making it asynchronous will invariably degrade the performance of that one operation, as it still needs to accomplish everything that the synchronous operation did, but now with additional constraints and considerations. A reason you care about asynchronicity, then, is performance in the aggregate: how your overall system performs when you write everything asynchronously, such that you can overlap I/O and achieve better system utilization by consuming valuable resources only when they're actually needed for execution. The asynchronous method implementation provided by the .NET Framework is well-optimized, and often ends up providing as good or better performance than well-written asynchronous implementations using existing patterns and volumes more code. Any time you're planning to develop asynchronous code in the .NET Framework from now on, asynchronous methods should be your tool of choice. Still, it's good for you as a developer to be aware of everything the Framework is doing on your behalf in these asynchronous methods, so you can ensure the end result is as good as it can possibly be. ■

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, Eric Lippert, Danny Shih and Mads Torgersen

CREATING EDITING PRINTING CONVERTING REPORTING ? IMPORTING or EXPORTING ?

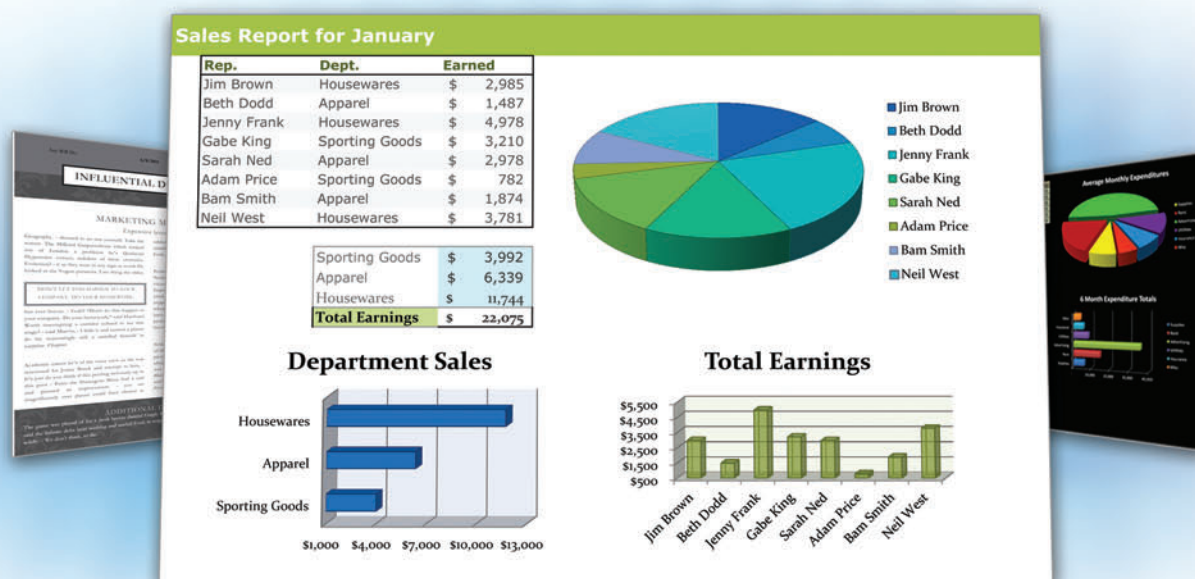
Documents Spreadsheets Presentations Barcodes Email

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

Securing Access to LightSwitch Applications

Valerie G. Andersen, Matt Evans, Sheel Shah and Michael Simons

Let's face it: Implementing application security can be daunting. Luckily, Visual Studio LightSwitch makes it easy to manage permissions-based access control in line-of-business (LOB) applications, allowing you to build applications with access-control logic to meet the specific needs of your business.

A LightSwitch application logically consists of three tiers: presentation, logic and data storage, and you'll need to consider access to assets at each tier in order to ensure the right level of access is achieved. With LightSwitch, you can build access-control logic into applications at the right level. Moreover, you'll find that LightSwitch leverages the access-control fundamentals in the underlying technologies and allows for common access-control configuration through IIS and ASP.NET.

This article examines how access control works in LightSwitch applications. First, we'll describe the features LightSwitch provides for access control in a three-tier architecture. Next, we'll briefly review deployment as it pertains to access control and show some ways to further control access using the technologies that support LightSwitch. Finally, we'll discuss access control when deploying to a Windows Azure environment.

This article discusses:

- Authentication and authorization
- Securing access in three-tier applications
- Controlling access across relationships
- Leveraging access-control features in IIS
- Securing LightSwitch Azure applications

Technologies discussed:

Visual Studio LightSwitch, Windows Azure, IIS, ASP.NET

Access Control Basics

There are two aspects to access control in LightSwitch applications. The first is *authentication*, or how an application verifies a user is who he says he is. The second is *authorization*, which defines what the user is allowed to do or see in the application.

Authentication

The authentication process determines if a user is who he claims to be. The first layer of access control in LightSwitch requires users to identify themselves to the application. The supported authentication modes are *Windows authentication* and *Forms authentication*. These options can be configured in the application properties Access Control tab of your application, as seen in **Figure 1**.

Windows authentication is recommended when all users are on a Windows domain and you trust that the person logged into a computer is the same user who's using the application. Windows authentication doesn't require an additional login prompt and the application doesn't have to store or manage passwords outside of Windows. Windows authentication is the more secure option, but it's usually only practical if the application is running in a corporate intranet environment with a domain.

With the second option, Forms authentication, the user is prompted for a username and password when the application opens. In LightSwitch, these values are checked against a database by default. Forms authentication works nicely for clients running across the Internet or for those not on a Windows domain.

Forms authentication requires that any users needing access to the application first be added to the system. Windows authentication can work this way as well, but there's an option that can be set at design time to allow all Windows users who can log in to the domain to have access to the application by default. Any parts

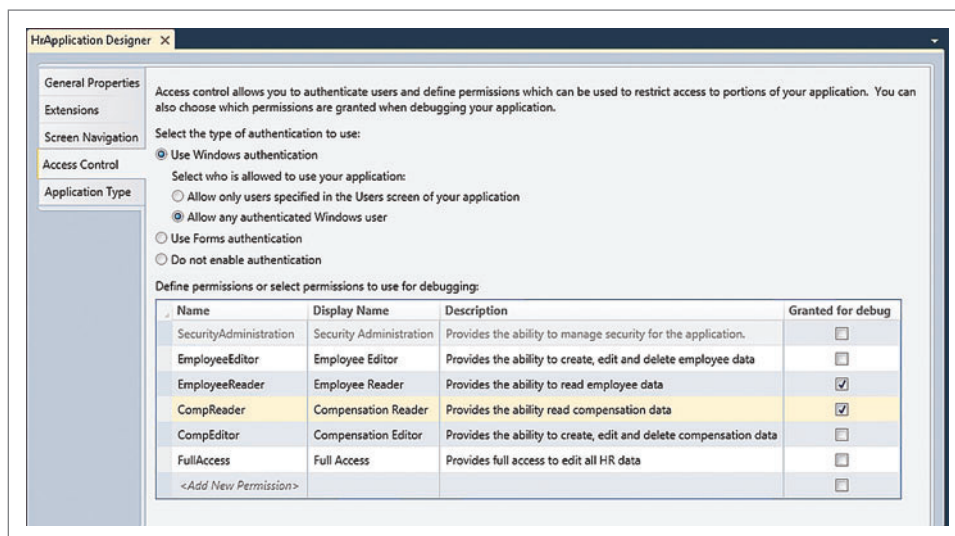


Figure 1 Defining Permissions in the Application Designer

of the application requiring specific permissions would not be accessible to a Windows user who hasn't been explicitly added.

Authentication lets you identify who can or can't use the application, and it may be all that's required to meet the access-control needs for some kinds of applications. Once users are authenticated, you may choose to fully trust them with access to the data. In that case, your access-control implementation is complete and no additional permissions or code are required. You need only consider the IIS options discussed in the Deployment Considerations for Access Control section for securing your application on the hosting server.

However, many applications require more granular control of users' behavior after they've been authenticated. For these scenarios, you'll need authorization.

Authorization

LightSwitch provides a permissions-based authorization system for developing business rules, as shown in **Figure 2**.

You define permissions in the application designer (see **Figure 1**). You can then write code to check if the current user has the required permissions. Access-control methods can be implemented on entities, queries and screens, so you can easily write logic to determine if the current user can view or manipulate specific data or open a particular screen.

LightSwitch has a built-in permission called Security Administration, and any user who receives this permission becomes a security administrator. The Security Administration permission allows the logged-on user to access the Security Administration screens in the running LightSwitch client application, which will show automatically for users who have been granted this privilege. The security administrator creates the roles needed for the application and assigns the desired permissions to each role, as shown in **Figure 3**. Then, users are created and assigned to the appropriate role, as shown in **Figure 4**.

When an application is first deployed, a security administrator must be added to the Security Administration role to enable initial access to the application. The deployment process assists in configuring

this default user appropriately. When a deployed application is running, the system will not allow the removal of the last user having the security administrator permission to ensure that a security administrator exists at all times.

However, you won't have to deploy your application to verify the appropriate permission behavior. When you run a LightSwitch application in debug mode, the authentication and role system run in a special mode in which the development environment automatically tries to authenticate a special test user. You can grant or deny the permissions the test user has from the LightSwitch

designer using the "Granted for debug" column. Then the application will run in debug mode with the permissions selected so you can test the written permission logic. This means you can quickly validate that the permission checks are accurate without having to configure multiple test users and roles.

Where You Control Access to Application Assets Matters

Now let's consider what to secure and where to secure it. Many applications have some sensitive data that needs to be protected from view, other data that only needs to be guarded against manipulation, and possibly some data that doesn't need protected access at all.

The first place to think about securing data should be the logic tier. When developers control data appropriately at the logic tier, the presentation tier will often react correctly and automatically. For example, if a user doesn't have the delete permission granted for Employee data, a grid showing Employee data will have the delete button disabled. That's the power and ease of building applications with LightSwitch.

Implementing access control at the data level makes the application more secure and lets developers take advantage of built-in intelligence. Securing data only at the presentation tier leaves your data exposed at the logic tier. A malicious, authenticated user could bypass the presentation tier and access the service directly to read or manipulate data. This relates to the three-tier architecture of LightSwitch applications and, in fact, is common among three-tier applications. The presentation tier is responsible for displaying the data appropriately; however, it's not the only way for an authenticated user to access the data if the proper access controls aren't implemented on the logic tier.

Securing Data on the Logic Tier

The logic tier includes two major groups of components where you apply permissions checking: entities and queries.

Entities Entities are the general mechanism for accessing and working with application data. There are four main actions you can

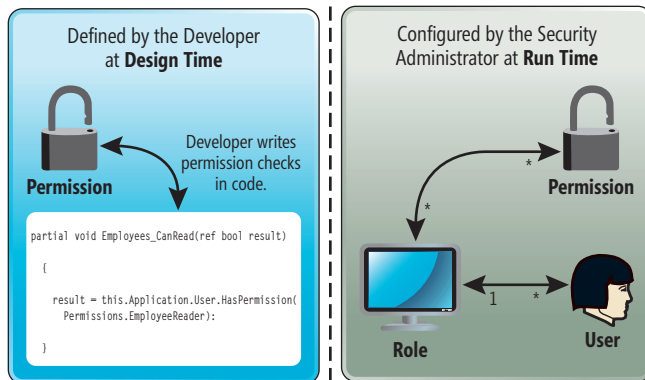


Figure 2 Implementing Authorization in LightSwitch

carry out with entities: read, insert, update and delete. LightSwitch gives developers a hook-point to verify a user's permissions as each action is carried out, and it also provides a simple API for checking if the current user has a specific permission defined in the application. The following code shows an example of the permissions check and the various gate methods the API provides to allow a developer to check permissions:

```
partial void Employee_CanUpdate(ref bool result)
{
    result = Application.Current.User.HasPermission(Permissions.EmployeeUpdate);
}

partial void Employee_CanInsert...

partial void Employee_CanRead...

partial void Employee_CanDelete...

partial void SaveChanges_CanExecute...
```

A couple of things should be noted. First, these methods, with the exception of `SaveChanges_CanExecute`, are implemented on

the entity set as a whole and not on the specific entity instances. Therefore, any checks performed can't be related to data values on a specific entity instance. The `SaveChanges_CanExecute` method controls access to changes in the entire data source and therefore can't contain entity- or entity-set-specific logic. Second, if the `Employee_CanRead` method returns false, the `Employee_CanUpdate` and `Employee_CanDelete` methods will not be called, as they would be implicitly false as well. A user is not allowed to update or delete an entity if she isn't allowed to read it.

The "Can" methods are the basic way to do coarse-grained security. They support basic data access-control policies. However, they have some limitations. When more fine-grained control for reading data is needed, you can implement access-control logic on queries. To control the writing of data at a more granular level, you must do so in the Save Pipeline.

Queries Queries are also secured on the logic tier. Each query has a method that allows you to control access. LightSwitch automatically generates three queries for each entity that exists: an All query to return all of the entity instances, and Single and SingleOrDefault queries to return one entity instance by key. Each of these built-in queries has a `CanExecute` method that can be used to implement access control:

```
partial void Employees_All_CanExecute(ref bool result)
{
    result = Application.Current.User.HasPermission(Permissions.QueryEmployees);
}

partial void Employees_Single_CanExecute...

partial void Employees_SingleOrDefault_CanExecute...
```

It's important to note that LightSwitch queries are composable, meaning new queries can be based on an existing query. When applying access-control logic to queries, the permission requirements on the initial query serve as input to queries composed on that query. The Single and the SingleOrDefault queries are composed

on the All query, so securing the All query also secures these queries if no specific permissions are specified for the derived query. However, if you like, you can specify permissions on the derived query that are less restrictive than those on the composing query. Also, the `CanRead` method on the entity set will be applied before `CanExecute` for any queries of that type.

Figure 5 shows an example of query composition—if the `NorthAmericaEmployees` query is created on the `Employee` entity, this query is composed on the built-in `Employee_All` query. Therefore, any access-control logic applied via `Employee_All_CanExecute` will also apply to the `NorthAmericaEmployees` query because the `NorthAmericaEmployees` query is based on the

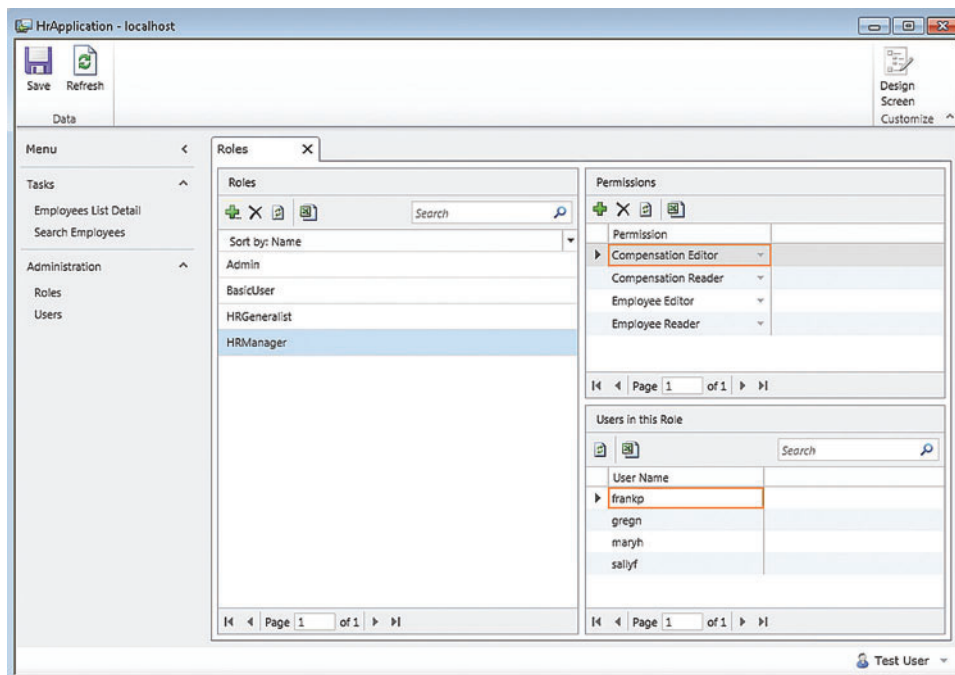


Figure 3 Creating Roles at Run Time

Employee_All query, assuming no specific code is written for the derived query. If you wanted to allow only certain users to access the data in the NorthAmericanEmployees entity, you could specifically restrict or open up the permissions for that query via the NorthAmericaEmployees_CanExecute method.

Controlling Access Across Relationships

When working with queries across relationships, it's important to understand that permissions are checked on the entities as relationships, which are traversed across related entities. This is another reason why it's important to have permissions properly defined on the entities. If you need to protect data from read access through relationships, the CanRead method on the entity needs to require the correct level of permission. As an example, let's consider our Employee table again and the related compensation data, as shown in the model in Figure 6.

When traversing the relationship between Employee and Compensation via a query, the permissions on the entity read actions are evaluated as the relationship is traversed rather than the permissions on the Compensation_All_CanExecute. The permissions must be correctly set on the Compensation entity's CanRead method so the correct level of permission is achieved as entities are traversed. You need to be aware that queries can be used to infer data when the entities are not secured. For example, if you have a query that returns the top-paid employees, as shown in Figure 7, the compensation entity that's accessed to return the Employee data needs to be properly secured so only users who have been granted access can get to this data via the query.

Providing a Customized Presentation Experience

Once the data has been secured on the logic tier, it's time to design the user experience in the presentation tier. If you don't want a user to have access to a particular screen at all, you can turn the screen off via the <ScreenName>_CanRun method for the application. When a user doesn't have access to a given screen, the screen won't show in her navigation menu. You can also restrict commands on a screen by using the <CommandName>_CanExecute method.

There are several other methods available on screens and entities that can be used to hide, show and control the editable state of specific controls on a screen, such as

<EntityProperty>_IsReadOnly, <ScreenControl>.IsEnabled, <ScreenControl>.IsReadOnly and <ScreenControl>.IsVisible. While the main purpose of these methods is not access control, they're quite useful in delivering the desired user experience. In some cases, it might be best to hide data users can't manipulate; other times you may want to show read-only controls; and sometimes you'll want to guide users to enter data correctly and give meaningful errors if they go astray. The LightSwitch presentation tier allows all this flexibility.

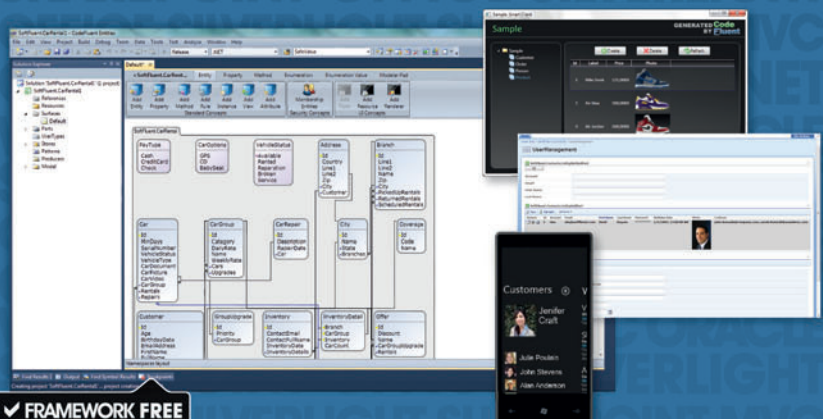
Less Plumbing Code, More Features use CODEFLUENT ENTITIES!

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.



- ✓ 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



Contact: info@sofflulent.com
Twitter: twitter.com/sofflulent
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.

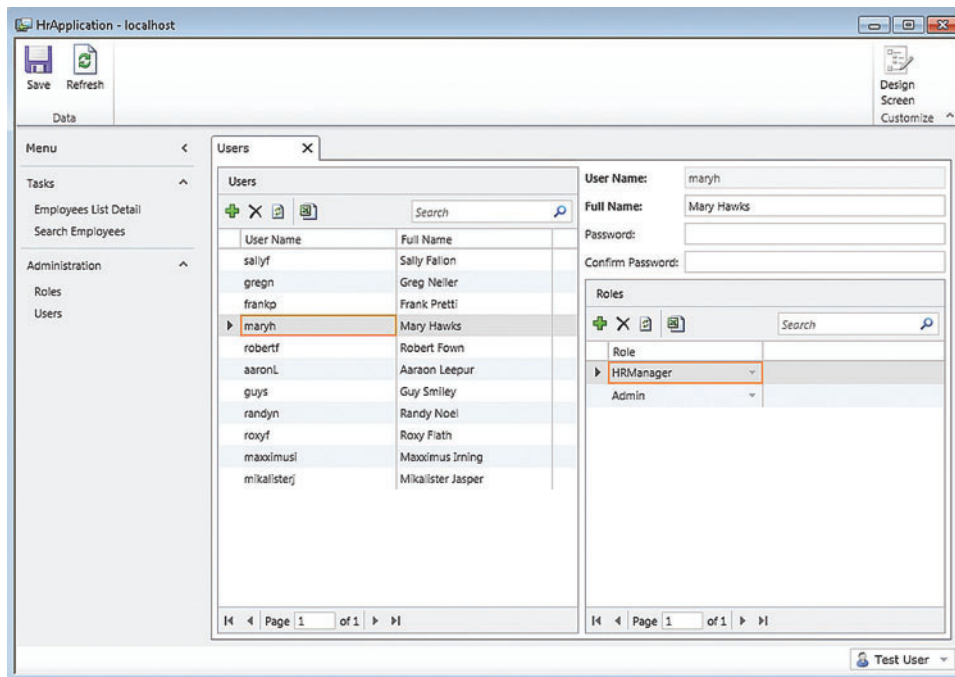


Figure 4 Assigning Users to Roles at Run Time

It should be clearly understood that *providing logic on the screen to hide, show or control the editable state of data does not protect data from user access*; it only controls how the data is displayed. A malicious, authenticated user could potentially hit the logic-tier service directly to view or manipulate data if it's not secured appropriately. That's why it's vital to implement access control on each application tier appropriately.

Securing Data on the Storage Tier

The storage tier is where your data is stored in a database. You can control access to the data in the storage tier by providing a database login with the minimum necessary privileges on the database. The application uses this login to connect to the database and carry out the necessary operations. All connectivity to the data is done via the middle tier, and end users never have direct access to the data or connection string in a three-tier deployment. When you're deploying a LightSwitch application, you must specify a connection string that the application will use to access the data, as shown in **Figure 8**. If a unique database login doesn't exist for the application, LightSwitch will guide the application administrator to create one. It's highly recommended that the database user identified be specific to the application and be given access only to the pertinent data used by the application.

It's worth mentioning that it's possible to deploy a secure LightSwitch application with a two-tier deployment. In a two-tier deployment, the presentation tier and the logic tier run on the user's desktop. This configuration gives the user of the client access to the web.config file where the connection string for the database is stored, therefore it does not offer the separation of presentation and logic tiers required

to achieve a secure application configuration. With the connection string, the user can gain direct access to the database and bypass any access-control logic on the middle tier. It's best to use Windows authentication between the middle tier and the database in this case. Otherwise, a three-tier deployment is necessary to secure the application appropriately.

Deployment Considerations for Access Control

When deploying a LightSwitch application for the first time, you need to create a LightSwitch administrative user. Initially, this is the only user who will have administrator permissions. This user will then be used to configure roles and users within the

client application as discussed earlier. Note that this user will be an administrator of your LightSwitch application, but need not be a Windows administrator. A command-line utility is also available to create a new administrative user outside of the deployment process. You can find this utility, Microsoft.LightSwitch.SecurityAdmin.exe, in the LightSwitch install directory.

Leveraging Access-Control Features in IIS

Now that we've discussed LightSwitch-specific access-control functionality, let's briefly touch on some additional ways an application administrator can manually secure LightSwitch applications using the supporting technologies.

LightSwitch and SSL Like Web browsers and Web servers, LightSwitch clients and servers communicate via the HTTP protocol. HTTP specifies that data be sent in clear text, which means that a network eavesdropper could monitor the data the clients and servers exchange. To secure communications against network eavesdroppers, you should instead use the HTTPS protocol, which hides the normal HTTP data in an encrypted tunnel.

LightSwitch applications can be deployed such that clients communicate with the server via HTTPS. This protects not only the sensitive business data that's exchanged between client and server, but also usernames and passwords when using Forms authentication. It's a best practice to deploy to an HTTPS site when using Forms authentication in conjunction with IIS or Windows Azure. Otherwise, it's possible for an attacker to steal the authentication token and impersonate the logged-in user. When using Windows authentication, an attacker can't recover user passwords or impersonate users, even when using HTTP instead of HTTPS. However, regardless of

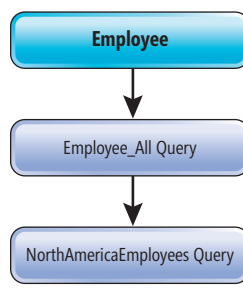
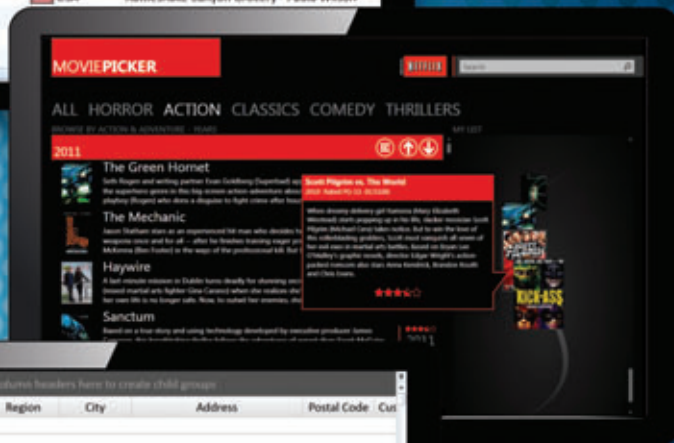


Figure 5 Query Composition on the Employee Entity

THESE GUYS STAND ON THEIR OWN.

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

ID	Employee	Country	Customer
USA (122 items)			
Albuquerque (18 items)			
11,077	Davolio, Nancy	USA	Rattlesnake Canyon Grocery - Paula Wilson
11,000	Fuller, Andrew	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,988	Leverling, Janet	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,889	Dodsworth, Anne	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,852	Callahan, Laura	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,820	Leverling, Janet	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,761	Buchanan, Steven	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,598	Davolio, Nancy	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,569	Buchanan, Steven	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,564	Peacock, Margaret	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,479	Leverling, Janet	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,401	Davolio, Nancy	USA	Rattlesnake Canyon Grocery - Paula Wilson
10,346	Leverling, Janet	USA	Rattlesnake Canyon Grocery - Paula Wilson



Order ID	Country	Region	City	Address	Postal Code	Customer
USA						
CO						
FL						
10310	USA	FL	Jacksonville	89 Jefferson Way Suite 2	97201	77
10317	USA	FL	Jacksonville	89 Chiaroscuro Rd.	97219	48
10805	USA	FL	Miami	89 Jefferson Way Suite 2	97201	77
10867	USA	FL	Jacksonville	89 Chiaroscuro Rd.	97219	48
10883	USA	FL	Miami	89 Chiaroscuro Rd.	97219	48
10992	USA	FL	Jacksonville	89 Jefferson Way Suite 2	97201	77
11018	USA	FL	Jacksonville	89 Chiaroscuro Rd.	97219	48
11169	USA	FL	Miami	89 Chiaroscuro Rd.	97219	48
11172	USA	FL	Jacksonville	89 Jefferson Way Suite 2	97201	77
11179	USA	FL	Miami	89 Chiaroscuro Rd.	97219	48
11406	USA	FL	Miami	89 Chiaroscuro Rd.	97219	48
11524	USA	FL	Miami	89 Chiaroscuro Rd.	97219	48
11729	USA	FL	Jacksonville	89 Chiaroscuro Rd.	97219	48
11854	USA	FL	Jacksonville	89 Jefferson Way Suite 2	97201	77
11880	USA	FL	Miami	89 Chiaroscuro Rd.	97219	48



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

XCEED
MULTI-TALENTED COMPONENTS

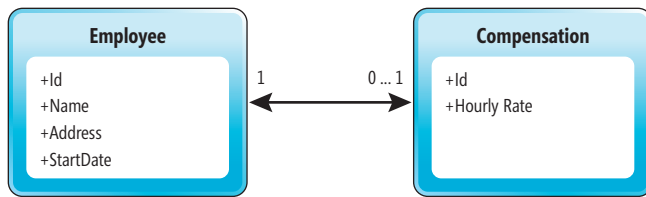


Figure 6 HR Application Model

authentication mode, business data transferred between client and server is still subject to eavesdropping unless HTTPS is employed.

SSL and Certificates Web servers that communicate via HTTPS rely on a server certificate being installed. The certificate serves two purposes. First, the certificate verifies that the server the client is connecting to is actually the correct server and hasn't been replaced or tampered with. Second, the server certificate contains the secret key information used to encrypt any data sent to the client.

For a Web browser to be able to trust the identity of a server it hasn't contacted previously, the server's certificate must have been cryptographically signed by a trusted certificate authority (CA). You can purchase a certificate from a number of providers, such as verisign.com, entrust.net, instantssl.com or geocerts.com. Because most providers charge to generate or sign server certificates, it's common in development environments to use a self-generated, unsigned—and thus untrusted—certificate.

Web servers that communicate via HTTPS rely on a server certificate being installed.

If you connect to a LightSwitch Web application via HTTPS and the server is using an untrusted certificate, the behavior depends on your Web browser. Minimally, your browser will inform you of the certificate problem and ask if you'd like to proceed. For LightSwitch Web applications, this should work correctly.

However, if you're using a LightSwitch desktop application that's hosted by IIS, and accessing it via HTTPS, the IIS server must be using a trusted certificate. Silverlight will not allow desktop applications to come from untrusted server certificates. An untrusted certificate will result in an application that appears to install successfully but will fail immediately upon launch. To remedy this problem, either force your Web browser to trust the server's certificate by pre-installing it in the client's certificate store or replace the server's certificate with one that has been signed by a trusted CA. Note that you'll want to perform

one of these corrective steps before accessing the application for the first time from a given client; otherwise, it will appear to that client as though the certificate has changed or been tampered with.

IIS No LightSwitch-specific configuration is required on the IIS server in order to host LightSwitch applications with SSL. Server administrators should configure the Web server to enable HTTPS, and select a certificate in the normal fashion.

It's actually possible to host the same LightSwitch application using both HTTP and HTTPS, and there are some situations where you might want to do this. But note that, as already mentioned, any clients that connect via HTTP are not protecting user password information or sensitive business data.

By default, in recent versions of IIS, the Default Web Site listens for both HTTP and HTTPS connections. The server administrator can force a LightSwitch application deployed to such a server to require HTTPS and redirect any HTTP incoming connections to the HTTPS endpoint. This setting is in the `web.config` of the LightSwitch application.

Application Pool Configuration When publishing to IIS, consider running the application in its own application pool. Application pools provide isolation between worker processes (so if one Web application crashes it doesn't affect other applications on the server), but they also allow the application to run under different identities. Thus you can create an application pool that hosts a Web application or a set of services that runs under a specific Windows identity and allow access only to the resources that identity needs to run the application. In the case of a LightSwitch application, that additional resource will be the database. By default, the deployment wizard publishes the application under the ASP.NET 4 application pool that runs under a machine account identity. However, this identity doesn't have access to the application database, so running the application results in an error message such as "Login failed for user 'IIS APPPOOL\ASP.NET v4.0.'"

There are a couple of options here. If you're using a SQL Server username/password in the connection string, the application is most likely ready to go (as long as that user had appropriate access to the database). However, when Windows authentication is preferred to connect to the database, a separate application pool is necessary so it can be configured to run under a least-privileged Windows user account. This account must also be granted appropriate access to the application database.

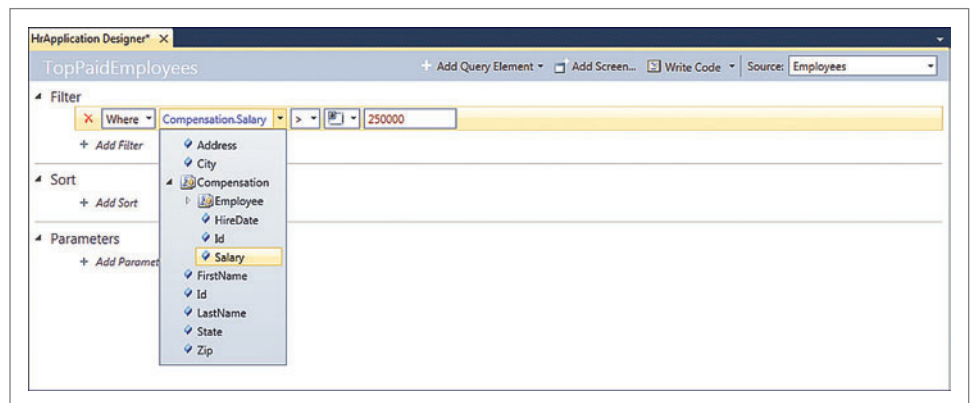
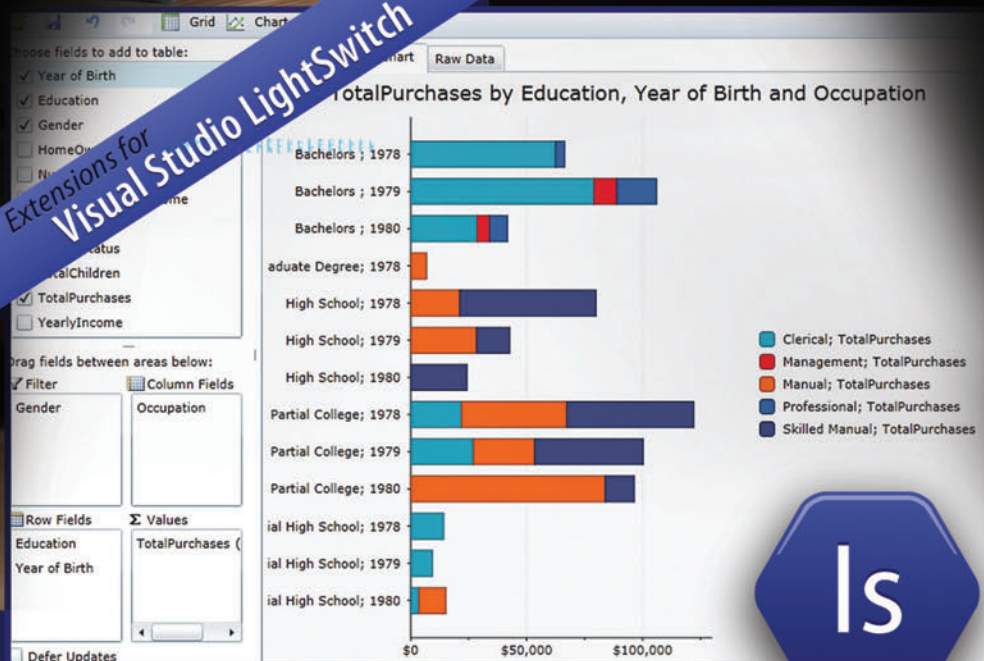


Figure 7 Defining a Query to Return the Top-Paid Employees

BUSINESS INTELLIGENCE *in the Flip of a Switch*

Extensions for
Visual Studio LightSwitch



componentone.com/lightswitch

 **ComponentOne**

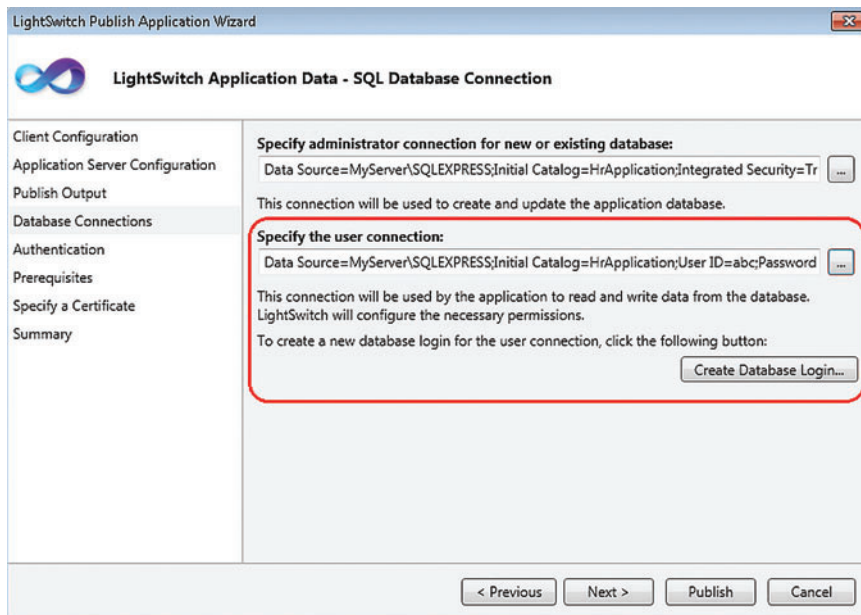


Figure 8 Specifying the Database Connection Credentials for the Running Application During Deployment

If you're using Windows authentication in IIS 7, there's one additional configuration setting of which you should be aware. When you use a custom user identity as the application pool identity, you need to either use the Windows NT LAN Manager (NTLM) provider (not Kerberos) or enable support for Kerberos authentication. Another option is to use the Network Service identity as the app pool's identity and add that account access to the database instead.

Securing LightSwitch Azure Applications

All LightSwitch applications that are hosted in Windows Azure are publicly accessible. Therefore, these applications have a unique set of access-control requirements.

SSL Encryption LightSwitch will default to using an HTTPS endpoint when publishing an application to Windows Azure. This is to ensure that any sensitive business information is encrypted as it's communicated over the Internet. LightSwitch provides an option to create a self-signed SSL certificate for the application during publish. While this is a great way to test the application in Windows Azure, it's highly recommended that a licensed certificate from an external vendor be used, as noted earlier. Because desktop applications won't work over SSL with an untrusted certificate, you can turn SSL encryption off for debugging purposes by updating the deployment configuration value of `Microsoft.LightSwitch.RequireEncryption` to false, using the Windows Azure portal to do so after the application has been successfully deployed.

Once the application has been tested using a self-signed SSL certificate, you can update the SSL certificate without republishing the application via the Windows Azure portal. A new SSL certificate can be uploaded for the hosted application by changing the `SSLCertificate` value to the thumbprint for the newer certificate and turning encryption back on.

Application Authentication Forms authentication is recommended to prevent unauthorized access to Windows Azure-hosted

LightSwitch applications. This requires no additional server configuration after an application has been published. If the application requires Windows authentication, though, the published LightSwitch application will need to be domain-joined. This requires the use of Windows Azure Connect. You'll find guidance on enabling Windows Azure Connect at bit.ly/qx0Z6n.

SQL Azure Database Security LightSwitch applications will typically rely on a SQL Azure database for their built-in database. This database is necessary if you've created any tables in LightSwitch or are using authentication. SQL Azure uses the combination of a firewall and user credentials to protect against unauthorized access.

To allow Windows Azure-hosted LightSwitch applications to connect to the database, the "Allow other Windows Azure services to access this server" firewall rule must be set to true. LightSwitch also requires that a fire-

wall rule be added for the IP address of the machine publishing the application. It's recommended that this firewall rule be removed once the application has been published. This will prevent any external machines from accessing the database.

Wrapping Up

LightSwitch helps developers build business applications in a simple and productive manner, and this holds true for implementing access-control features as well. Developers can quickly and easily restrict access to their applications through the use of authentication. When more granular control is needed, authorization features provide a powerful way to define permissions and secure assets at the logic and data tiers of the application to effectively control user access. Commonly used features in IIS and Windows Azure can be leveraged for a full access-control solution. The innovation, however, is up to you! Look for more from the LightSwitch team at blogs.msdn.com/b/lightswitch. ■

VALERIE ANDERSEN is a Microsoft program manager working on Visual Studio LightSwitch. Her aim is to drive features into LightSwitch that will enable real-world developers to swiftly build secure, quality applications to meet the needs of customers worldwide.

MATT EVANS is a software tester working on LightSwitch. He wants to make sure that your LightSwitch apps are as secure as you need them to be.

SHEEL SHAH is a Microsoft program manager working on LightSwitch. His focus on the team includes designing Windows Azure support, deployment and LightSwitch client features.

MICHAEL SIMONS is a senior Microsoft developer working on LightSwitch. His focus on the team is developing data and security features.

THANKS to the following technical experts for reviewing this article: Dan Leaphon, John Rivard, Dan Seefeldt and Matt Thalman

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 files 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



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

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

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





#1 PRODUCT
ComponentSource Awards 2010-11



#1 PUBLISHER
ComponentSource Awards 2010-11



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

DevExpressTM

Authoring an F#/C# VSIX Project Template

Daniel Mohl

Software developers are increasingly expected to provide complex solutions in less time and with fewer defects. One area that often causes productivity loss is the initial setup of new F#, C# or Visual Basic projects in just the way you require. The best way to reduce these repetitive project setup tasks is to create a Visual Studio Extension (VSIX) project template.

You may also wish to author a VSIX project template to promote software development standards, showcase a new product or provide a quick launch into a new technology. Whatever your reason, adding the ability to create these VSIX project templates to your development bag of tricks is certain to have benefits.

In this article, I'll show you how to author a VSIX project template made up of a C# ASP.NET MVC 3 Web Application, an F# Library

that contains the server-side code and an F# Library that can be used to contain unit tests. Additionally, you'll learn a few advanced techniques that can be used to add flexibility and power to your project templates. In the end, you'll be armed with the knowledge to greatly reduce the previously mentioned time-wasters through the creation of your own custom project templates.

Getting Set Up

Before diving in, you must complete a few tasks in order to get your development environment set up. First, make sure you have Visual Studio 2010 Professional or higher with the F# and C# components installed. Next, install the Visual Studio 2010 SDK, which you can download from bit.ly/vs-2010-SDK. The Visual Studio 2010 SDK provides everything needed to create a VSIX project. To help reduce the tedium associated with creating a multiproject template, you should also download and install the Export Template Wizard from bit.ly/export-template-wiz. Finally, in order to complete the example provided with this article, you should download and install the ASP.NET MVC 3 Tools Update, available at bit.ly/mvc3-tools-update. This update provides a number of useful features and tools, including NuGet 1.2. For more information, check out bit.ly/introducing-mvc3-tools-update.

Building the Initial Application

Now that your environment is set up, you're ready to build the base application that will be created every time the project template is launched from the New Project Wizard within Visual Studio. This lets you set up exactly what you need in order to

This article discusses:

- Setting up your environment
- Building the base application
- Creating a VSIX project template
- Extending the template
- Adding a Windows Presentation Foundation UI

Technologies discussed:

F#, C#, ASP.NET MVC 3

Code download available at:

fsharpmvc3vsix.codeplex.com

reduce time wasted by repetitive project-initialization tasks. The solution developed at this point can be as simple or complex as needed.

The first project you need to create is a C# ASP.NET MVC 3 Web application that should be named MsdnWeb. The role of this project is to provide the presentation layer for the overall solution. You should launch the New Project Wizard in Visual Studio and select ASP.NET MVC 3 Web Application, which can be found within Visual C# | Web. For this example, you should ensure that the “Empty” template, the Razor view engine and the “Use HTML5 semantic markup” options are selected, and then click OK. Once Visual Studio finishes generating the project, locate the Global.asax and Global.asax.cs files. You’ll be writing the majority of the code for the Global.asax in F#, so you can now remove the Global.asax.cs file and update the Global.asax markup as shown in **Figure 1**. As a final step for this project, you can create a new folder in the Views folder named Home and add a new View within it named Index.

Next, create a new F# Library project named MsdnWebApp and remove the default .fs and .fsx files that Visual Studio creates. The primary purpose of this project is to contain the controllers, models and other server-side code that will drive the views. Because this is an F# project, you’ll be able to create this server-side code in the clean, crisp and succinct style that F# provides. In order to use this project for its intended purpose, you need to add references to the System.Web, System.ComponentModel.DataAnnotations, System.Web.Abstractions and System.Web.Mvc (version 3.0+) assemblies. You also need to add a Global.fs file that contains the code shown in **Figure 2**.

You may also wish to author
a VSIX project template to
promote software development
standards, showcase a new
product or provide a quick
launch into a new technology.

You also need to add a HomeController.fs file that contains the code shown here:

```
namespace MsdnWeb.Controllers
{
    Open System.Web
    Open System.Web.Mvc

    [HandleError]
    type HomeController() =
        inherit Controller()
        member this.Index() =
            this.View() :> ActionResult
}
```

As a final task in building the initial application, you should create the second F# Library project and name it MsdnWebAppTests. As

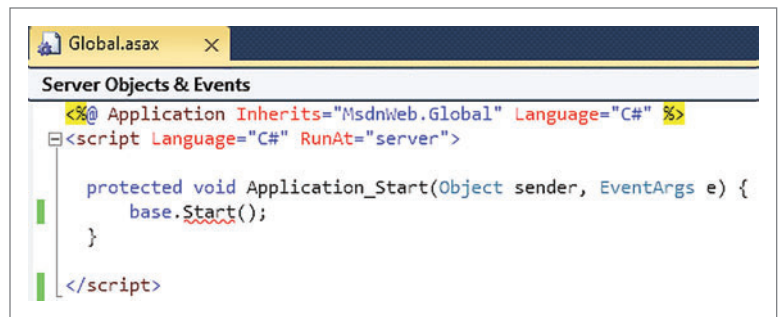


Figure 1 Updating the Global.asax Markup

the name suggests, the purpose of the MsdnWebAppTests project is to contain the unit tests for MsdnWebApp. A production-ready version of this project would be primed with unit tests. However, in order to keep things simple, you can remove the default .fs and .fsx files that Visual Studio generates and leave the empty project as a container for the creation of future tests. To see an example of the code that could be added to this project, install the FsUnit.MvcSample NuGet package at bit.ly/fsunit-mvc3-tests.

Creating the VSIX Project Template

The example project template I’m walking you through has several goals. These goals are the foundation on which the rest of this article is based:

1. Provide a VSIX project template made up of a C# project and two F# projects.
2. Dynamically add each of the projects to a shared solution during new project creation.
3. Programmatically add any necessary project references each time the project template is launched.
4. Install various NuGet packages to the created ASP.NET MVC 3 Web Application.
5. Provide a UI that allows a user to specify various project-creation options.

Visual Studio provides a rich extensibility model that makes accomplishing the first goal fairly trivial. The easiest way to take advantage of this is to launch the Export Template Wizard by going to File | Export Template as VSIX. Visual Studio will then display a VSIX package-creation wizard that allows the selection of multiple projects to be compressed into a VSIX package. While this handles simple multiproject scenarios, it can’t handle more advanced requirements such as goals two through five. To do these things, you’ll need a bit more power.

This power comes in the form of composition via the IWizard interface. This is commonly referenced as creating a template wizard. To build a simple template wizard in F#, you need to create a new F# Library project, add a class that implements the IWizard interface (see **Figure 3**) and add references to EnvDTE and Microsoft.VisualStudio.TemplateWizardInterface.

In order to handle more-advanced functionality such as programmatically adding project references and installing NuGet packages, you’ll also need to add references to EnvDTE80, VSLangProj, VSLangProj80, Microsoft.VisualStudio.ComponentModelHost, Microsoft.VisualStudio.OLE.Interop,

Figure 2 The Global.fs File

```
namespace MsdnWeb

open System
open System.Web
open System.Web.Mvc
open System.Web.Routing

type Route = { controller : string
              action : string
              id : UrlParameter }

type Global() =
    inherit System.Web.HttpApplication()

    static member RegisterRoutes(routes:RouteCollection) =
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}")
        routes.MapRoute("Default",
            "{controller}/{action}/{id}",
            { controller = "Home"; action = "Index"
              id = UrlParameter.Optional } )

    member this.Start() =
        AreaRegistration.RegisterAllAreas()
        Global.RegisterRoutes(RouteTable.Routes)
```

Microsoft.VisualStudio.Shell, Microsoft.VisualStudio.Shell.Interop, Microsoft.VisualStudio.Shell.Interop.8.0, NuGet.Core (version 1.2+) and NuGet.VisualStudio (version 1.2+). If you've followed the instructions for setting up your environment, all of these libraries are available on your local machine. A few of the key libraries can also be found in the lib folder of the source code that accompanies the article, which can be found at fsharpmvc3vsix.codeplex.com.

Because this is an F# project,
you'll be able to create this
server-side code in the clean,
crisp and succinct style that
F# provides.

As a final step to building the basic template wizard, you'll need to sign the template wizard assembly. If you don't already have one, you'll first need to generate a strong named key (.snk) file (see bit.ly/how-to-create-snk for information on how to do this). Once you have an .snk file, you can sign the F# template wizard assembly by going to the properties of the project, selecting the Build tab and typing the following into the "Other flags:" textbox (you must replace <Path> and <snk File Name> with the values that relate to your specific .snk file):

```
--keyfile:"<Path>\<snk File Name>.snk"
```

In order to use this template wizard, you'll need to create a multiproject template that references the signed template wizard assembly. The easiest way to do this is to use the Export Template Wizard Visual Studio 2010 extension to kick-start the process, then manually tweak the result. In the wizard that displays when you launch File | Export Template as VSIX, select the MsdnWeb,

MsdnWebApp and MsdnWebAppTests projects and click Next. On the resulting page, enter the template name and description that you wish to appear in the Visual Studio New Project Wizard window and specify the location of the template wizard DLL for the Wizard Assembly. After clicking Next, you should uncheck the "Automatically import the template into Visual Studio" option and click Finish. Visual Studio will then create the VSIX package, launch Windows Explorer and take you to the directory that contains the VSIX file.

This gives you a nice starting point, but now you must get your hands dirty and make a few manual modifications. Because VSIX files are really just .zip files with a different extension, you can dig into the guts of this package by simply changing the file extension to .zip and extracting all of the files. Once complete, navigate to the "Solution" directory within the folder that the extraction process revealed and extract the compressed file found within. This extraction process reveals the projects that constitute your project template as well as a .vstemplate file.

In order to meet the goals for this project template, you must modify this .vstemplate file. The first step you should do is rename the file to MsdnFsMvc3.vstemplate. Because this .vstemplate file is a simple XML file, you can now open the file in your text editor of choice and do the following:

1. Verify that the <Name> and <Description> elements contain the information that you wish to have displayed for this project template in the Visual Studio New Project Wizard.
2. Change the value in the <ProjectType> element to *FSharp*.
3. Remove all child elements of the <ProjectCollection> element.

You can now save and close the file, and then compress the three folders as well as the modified .vstemplate file into a file named MsdnFsMvc3.zip.

The MsdnFsMvc3.zip file could easily be combined back into a VSIX package at this point, but you would then be left without the ability to test or enhance the package with debugging support. It would be much better if Visual Studio could be used for these types of tasks. Luckily, the Visual Studio 2010 SDK that you previously installed provides exactly the type of tooling needed to do this. To get started, you first need to create a new VSIX Project, which can be found in the New Project Wizard under Visual C#

Figure 3 Implementing the IWizard Interface

```
namespace MsdnFsTemplateWizard

open System
open System.Collections.Generic
open EnvDTE
open Microsoft.VisualStudio.TemplateWizard

type TemplateWizard() =
    interface IWizard with
        member this.RunStarted (automationObject:Object,
                                replacementsDictionary:Dictionary<string,string>,
                                runKind:WizardRunKind, customParams:Object[]) =
            "Not Implemented" |> ignore
        member this.ProjectFinishedGenerating project = "Not Implemented" |> ignore
        member this.ProjectItemFinishedGenerating projectItem =
            "Not Implemented" |> ignore
        member this.ShouldAddProjectItem filePath = true
        member this.BeforeOpeningFile projectItem = "Not Implemented" |> ignore
        member this.RunFinished() = "Not Implemented" |> ignore
```



BEST SELLER



FusionCharts | from \$195.02

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

- Live 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 19,000 customers and 400,000 users in 110 countries

BEST SELLER



ActiveAnalysis | from \$979.02

GrapeCity PowerTools

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

BEST SELLER



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

Janus systems

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

BEST SELLER



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

TX CONTROL
word processing components

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

Figure 4 Filling out the Metadata

| Extensibility. The project that Visual Studio creates includes a `source.extension.vsixmanifest` file, which can be viewed and edited via a simple double-click. You can now fill out the basic metadata information about your project template. **Figure 4** provides an example.

The `MsdnFsMvc3.zip` file can now be added to the VSIX project in Visual Studio. To do this, you first navigate to the root folder of the VSIX project within Windows Explorer and create a new folder named `ProjectTemplates`. Within this folder, you should add a new folder named `ASPNET`. The creation of this second folder is what determines the project subtype for the project template. You now copy the `MsdnFsMvc3.zip` file into the `ASPNET` folder.

Back in Visual Studio, in the design view of the `source.extension.vsixmanifest` file, you should click the “Add Content” button. **Figure 5** shows the entries that you can specify in the resulting window.

The directory structure that’s automatically created in the VSIX project by this process now needs a little adjustment. You do this by right-clicking on the `ProjectTemplates` folder and creating a new folder named `ASPNET`. Last, you should move the compressed file from the `ProjectTemplates` folder to the `ASPNET` folder and click Yes when prompted to overwrite the existing file. Building the solution quickly reveals that the creation of your VSIX project template by Visual Studio has been successful. If desired, you could now add this project template to Visual Studio by double-clicking the `.vsix` file that was created by Visual Studio and walking through the resulting installation wizard.

Extending the Project Template

The steps completed so far accomplish the first goal for this template and set the stage for attaining the rest of the goals. These additional goals are primarily accomplished by adding code to the `RunFinished` method of the `IWizard` implementa-

tion of the template wizard. I’ll now walk you through the code needed to accomplish the remaining goals for this template.

The code required to accomplish the second goal of dynamically adding each of the projects to the solution during project creation is shown in **Figure 6**.

The first line of code in the method shown in **Figure 6** returns the location of the multiproject template on the machine from which the New Project Wizard was launched. The second defines a function named `AddProject`.

This function contains the code necessary to update the Visual Studio status bar, specifies the appropriate `.vstemplate` file that corresponds with the desired project template and adds the project from the template to the solution. The last lines of code call the `AddProject` function for the `MsdnWeb` and `MsdnWebApp` projects. A similar call to `AddProject` could’ve also been added for the `MsdnWebAppTests` project, if that were desired.

Visual Studio provides a rich extensibility model.

To accomplish the third goal of dynamically adding any desired project references, you need to first create a map of the project names and associated Project objects that make up the solution (see bit.ly/fsharp-maps for information on F# maps). This is accomplished by calling a custom function named `BuildProjectMap` with a provided argument of a collection of the Projects in the solution.

Figure 5 Adding Content in the Design View of a Manifest File



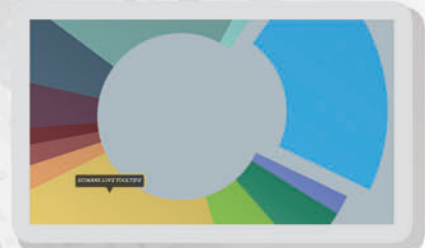
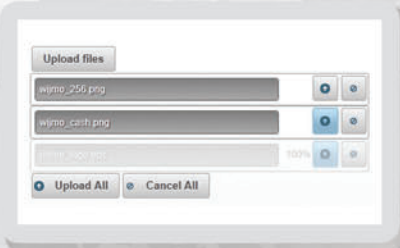
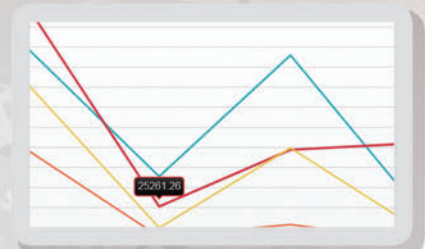
MEET THE NEW WEB STACK

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.



Personal Info					
#	Player	Age	Birthplace	Position	
27	Canada Adams, Craig	32	Serik, Sverdlovsk, Russia	R	R
43	Canada Bouchet, Philippe	36	Saint-Apollinaire, Quebec	D	R
24	Canada Cooke, Matt	30	Bellville, Ontario	LW	L
57	Canada Crosby, Sidney (C)	21	Scott's Hill, Nova Scotia	C	L
1	United States Curry, John	25	Shorewood, Minnesota	G	L
9	Canada Dupuis, Pascal	30	Laval, Quebec	W	L
7	United States Eaton, Mark	32	Wilmington, Delaware	D	L
26	Ukraine Fedchenko, Ruslan	30	Kiev, U.S.S.R.	LW	L
29	Canada Finlay, Marc-Alexis	24	Sorel, Quebec	G	L
15	Canada Giguere, Patrick	34	Shawinigan, Quebec	G	R



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

DOWNLOAD YOUR FREE TRIALS @
componentone.com/webstack

© 2011 ComponentOne LLC. All rights reserved. All product and company names herein may be trademarks of their respective owners.



This is then bound to a value named “projects,” as shown here:

```
// This function can be found in the full source
// as part of the ProjectService module.
let BuildProjectMap (projects:Projects) =
    projects
    |> Seq.cast<Project>
    |> Seq.map(fun project -> project.Name, project)
    |> Map.ofSeq

// This method can be found in the full source
// as part of the TemplateWizard class.
member this.RunFinished() =

    // Non-relevant code omitted.

    // this.dte is set in the RunStarted method.
    let projects = BuildProjectMap (this.dte.Solution.Projects)
```

// Non-relevant code omitted.

Now that you have the project map, you can call another custom function to kick off the process of adding the project references. The first line of the code shown here creates a list of tuples:

```
// This method can be found in the full source
// as part of the TemplateWizard class.
member this.RunFinished() =

    // Non-relevant code omitted.

    // webName, webAppName and webAppTestsName are values that have been
    // bound to strings that are used to identify specific projects.
    [(webName, webAppName); (webAppTestsName, webAppName)]
    |> BuildProjectReferences projects

    // Non-relevant code omitted.
```

Each tuple in the list represents the name of the target project followed by the name of the project that’s being referenced. For example, the first tuple indicates a target project name of “MsdnWeb” with an associated project reference name of “MsdnWebApp.” This list is then piped to the BuildProjectReferences function.

The process for accomplishing the final goal—adding a UI that can be used to gather information from the user during the project-creation process—hasn’t changed much over the past several years.

This code shows the BuildProjectReferences function:

```
// This function can be found in the full source
// as part of the ProjectService module.
let BuildProjectReferences (projects:Map<string, Project>) projectRefs =
    projectRefs
    |> Seq.iter (fun (target,source) ->
        AddProjectReference (projects.TryFind target)
        (projects.TryFind source))
```

This function simply takes the list of tuples, iterates through it, tries to retrieve the appropriate Project object from the project map by name and calls the AddProjectReference function to do the actual work.

Figure 6 Dynamically Adding Projects to the Solution

```
// This method can be found in the full source
// as part of the TemplateWizard class.
member this.RunFinished() =

    // Non-relevant code omitted.

    // this.solution is set in the RunStarted method.
    let templatePath = this.solution.GetProjectTemplate("MsdnFsmvc3.zip", "FSharp")
    let AddProject status projectVsTemplateName projectName =
        // this.dte2 is set in the RunStarted method
        this.dte2.StatusBar.Text <- status
        let path =
            templatePath.Replace("MsdnFsmvc3.vstemplate", projectVsTemplateName)
        this.solution.AddFromTemplate(path,
            Path.Combine(this.destinationPath, projectName),
            projectName, false) |> ignore

    // webName and webAppName are values that have been
    // bound to strings that identify specific projects.
    AddProject "Installing the C# Web project..."
        (Path.Combine("MsdnWeb", "MsdnWeb.vstemplate")) webName
    AddProject "Adding the F# Web App project..."
        (Path.Combine("MsdnWebApp", "MsdnWebApp.vstemplate")) webAppName

    // Non-relevant code omitted.
```

The AddProjectReference function finishes up the process by verifying that the target and projToReference arguments both contain a valid project, as shown here:

```
// This function can be found in the full source
// as part of the ProjectService module.
let AddProjectReference (target:Option<Project>)
    (projToReference:Option<Project>) =
    if ((Option.isSome target) && (Option.isSome projToReference)) then
        let vsTarget = target.Value.Object :?> VSProject
        vsTarget.References
        |> Seq.cast<Reference>
        |> Seq.filter(fun (reference) -> reference.Name = projToReference.Value.Name)
        |> Seq.iter(fun reference -> reference.Remove())
        vsTarget.References
        .AddProject((projToReference.Value.Object :?> VSProject).Project)
        |> ignore
```

If they do contain valid projects, this function removes any existing reference. Finally, it adds the reference to the project.

The fourth goal for this project template is a concept that was recently introduced by the ASP.NET MVC team. It provides a great way to add references to libraries or frameworks that are likely to be enhanced in the not-too-distant future. NuGet 1.2, which ships with the ASP.NET MVC 3 Tools Update, includes an assembly named NuGet.VisualStudio that allows NuGet packages to be installed easily from within a template wizard. The ASP.NET MVC 3 Tools Update install also adds several NuGet packages to the local machine to allow faster installation of these specific packages during project creation.

There are a couple of different functions in the sample that are used to accomplish the NuGet package installations. The most important one is show here:

```
// This function can be found in the full source
// as part of the NuGetService module.
let InstallPackages (serviceProvider:IServiceProvider) (project:Project) packages =
    let componentModel =
        serviceProvider.GetService(typeof<SComponentModel>) :?> IComponentModel
    let installer = componentModel.GetService<IVsPackageInstaller>()
    let nugetPackageLocalPath = GetNuGetPackageLocalPath()
    packages
    |> Seq.iter (fun packageId ->
        installer.InstallPackage(nugetPackageLocalPath,
            project, packageId, null, false))
```

The first three lines of code in this function are used to get the concrete implementation of the IVsPackageInstaller interface,

Visual Studio® **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

DECEMBER 5-9
ORLANDO, FLORIDA
ROYAL PACIFIC RESORT
AT UNIVERSAL ORLANDO®

5 CODE-FILLED DAYS IN SUNNY FLORIDA

- ▶ In-depth training for all levels of developers
- ▶ Impressive speaker lineup that includes top industry experts and Microsoft insiders
- ▶ 60+ sessions
- ▶ 8 tracks
- ▶ 5 full-day workshops
- ▶ Special events and networking opportunities

REGISTER TODAY: [VSLIVE.COM/ORLANDO](https://vslive.com/orlando)

Use promo code MTIP





**DECEMBER 5-9
ORLANDO, FLORIDA
ROYAL PACIFIC RESORT
AT UNIVERSAL ORLANDO®**

Sponsored by: **Microsoft**

INTENSE TRAINING + AN AWESOME LOCATION

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

**SAVE \$200!
REGISTER
TODAY**

Use promo code MTIP



Scan the
QR code
for more
information
on Visual
Studio Live!
Orlando.

VSLIVE.COM/ORLANDO

PLATINUM SPONSORS

Microsoft

SynCFusion
Deliver innovation with ease

SUPPORTED BY

Microsoft
Visual Studio

msdn

Visual Studio
MAGAZINE

PRODUCED BY

1105 MEDIA

which will be used to install the various NuGet packages in the specified project. The fourth line of code calls the `GetNuGetPackageLocalPath` function, which accesses the System Registry to determine the install path of ASP.NET MVC 3. Last, the provided list of package names is piped to the `Seq.iter` function, which iterates through the list and installs each package.

Now that all of this functionality has been implemented in your template wizard, you simply need to add the template wizard project as content to the VSIX project with a type of Template Wizard. **Figure 7** shows the completed Add Content window. This completes goals two through four.

Adding a Windows Presentation Foundation UI

The process for accomplishing the final goal—adding a UI that can be used to gather information from the user during the project-creation process—hasn't changed much over the past several years. A 2007 article from O'Reilly Media Inc. (oreil.ly/build-vs-proj-wiz) provides a nice overview for doing this. While the premise of the process remains the same, you need to know a few things to implement this functionality in Windows Presentation Foundation (WPF) and tie it into your project template.

An F#/C# VSIX project template is an easy way to encourage reuse.

To get started, you need to first create a new C# User Control Library. You should now change the target framework from .NET Framework 4 Client Profile to .NET Framework 4. Next, you can remove the default `UserControl1.xaml` and add a new WPF Window. Through whichever method you prefer, you can now manipulate the XAML to design the desired UI. Finally, you need to expose any desired properties and then define any necessary event handlers. A simple example of what the codebehind for the WPF Window might look like is shown here:

```
// This can be found in the full source as part of the MsdnCsMvc3Dialog class.

public bool IncludeTestsProject { get; set; }

private void btnOk_Click(object sender, RoutedEventArgs e)
{
    IncludeTestsProject =
        cbIncludeTestsProject.IsChecked.HasValue ?
        cbIncludeTestsProject.IsChecked.Value : false;
    DialogResult = true;
    Close();
}

private void btnCancel_Click(object sender, RoutedEventArgs e)
{
    Close();
}
```

After getting the UI exactly the way you want it, you'll need to sign the assembly. Next, you should add the project—as content with a Content Type of Template Wizard—to the VSIX project. Within the template wizard project, you then add a reference to the

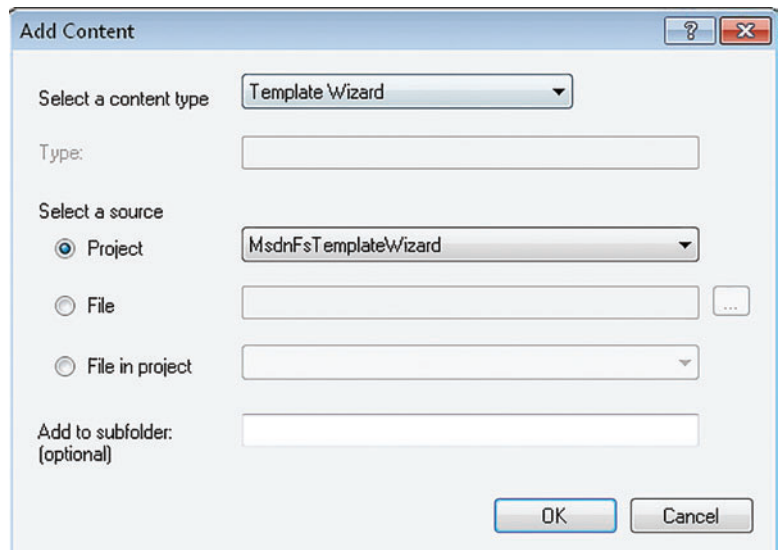


Figure 7 The Completed Add Content Window

project that contains your UI. Once this is all finished, you need to add code to the `RunStarted` method of the `IWizard` implementation that will show the UI and capture the result, as shown here:

```
// This method can be found in the full source
// as part of the TemplateWizard class.
member this.RunStarted () =

    // Non-relevant code omitted.

    let dialog = new TemplateView()
    match dialog.ShowDialog().Value with
    | true ->
        this.includeTestProject <- dialog.IncludeTestsProject
    | _ ->
        raise (new WizardCancelledException())
```

Finally, you can add the following code to the `RunFinished` method of the template wizard:

```
// This method can be found in the full source
// as part of the TemplateWizard class.
member this.RunFinished() =

    // Non-relevant code omitted

    // webAppTestsName is a value that has been bound
    // to a string that represents the tests project.
    if this.includeTestProject then
        AddProject "Adding the F# Web App Tests project..."
            (Path.Combine("MsdnWebAppTests",
                "MsdnWebAppTests.vstemplate")) webAppTestsName
    // Non-relevant code omitted.
```

Reuse and Reduce Time

Authoring an F#/C# VSIX project template is an easy way to encourage reuse and reduce time wasted on repetitive project set-up tasks. With these skills now in your possession, you'll be able to increase productivity by creating project templates that contain as little or as much complexity as your scenario requires. ■

DANIEL MOHL is a Microsoft MVP and F# Insider. He blogs at blog.danielmohl.com and you can follow him on Twitter at twitter.com/dmohl.

THANKS to the following technical experts for reviewing this article:
Elijah Manor, Chris Marinos and Richard Minerich

Harnessing the Power of the Dynamics CRM 4.0 API from Silverlight 4

Mark Beckner

Increasingly, companies are implementing Microsoft Dynamics CRM 4.0 solutions and finding it necessary to build external applications that can integrate with the existing Web service-based API.

Building Silverlight applications that can interact directly with Microsoft Dynamics CRM 4.0 (CRM 4.0 hereafter for brevity) can prove challenging, due to the asynchronous nature of Silverlight calls and its inability to call the CRM 4.0 Web services directly. In this article, you'll get an in-depth look at how to build a Silverlight application that can read and write data through the CRM 4.0 Web service API.

Solution Overview

Silverlight 4 and CRM 4.0 are both powerful technologies—but they aren't the most easily integrated. I'll look at the details behind

creating this integration by exploring asynchronous communications between Silverlight and the CRM 4.0 Web service API.

In typical non-Silverlight applications, a call to a Web service is synchronous—the call goes out, and the application waits until a response is received. During this time, the user can't interact with the application and must wait for the full roundtrip to complete. In an asynchronous application such as Silverlight, the call to the service goes out, but the application continues to be fully functional even before the response is returned. This creates a rich and dynamic user experience, but it places greater demands on the developer.

In order to understand how this communication can occur, I'll present several pieces of functionality. First, I'll look at how to set up a Silverlight application and work through the steps required to interact with a Web service that will act as a wrapper for the CRM 4.0 API. Next, I'll go through the details of working with the CRM 4.0 API and how to read and write data using the Web service wrapper. I'll work with the core System User entity in CRM 4.0,

This article discusses:

- Creating a Silverlight app
- Interacting with the Dynamics CRM 4.0 API
- Querying data
- Updating data
- Using returned data in Silverlight

Technologies discussed:

Silverlight 4, Microsoft Dynamics CRM 4.0

Code download available at:

code.msdn.microsoft.com/mag201110Dynamics

Figure 1 The Web Method Stubs

```
public class CrmServiceWrapper : System.Web.Services.WebService
{
    [WebMethod]
    public string GetCRMDData()
    {
        return "This is the stubbed return for retrieving";
    }

    [WebMethod]
    public string PostCRMDData()
    {
        return "This is the stubbed return for posting data";
    }
}
```

Figure 2 Adding Code to MainPage.xaml.cs

```
using CRM40SilverlightDemo.CrmServiceReference;

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

        // Call the GetCRMDData Web method.
        CrmServiceWrapperSoapClient proxyGet =
            new CrmServiceWrapperSoapClient();
        proxyGet.GetCRMDDataCompleted +=
            new EventHandler<GetCRMDDataCompletedEventArgs>(proxy_GetCRMDDataCompleted);
        proxyGet.GetCRMDDataAsync();

        // Call the PostCRMDData Web method.
        CrmServiceWrapperSoapClient proxyPost = new CrmServiceWrapperSoapClient();
        proxyPost.PostCRMDDataCompleted +=
            new EventHandler<PostCRMDDataCompletedEventArgs>(proxy_PostCRMDDataCompleted);
        proxyPost.PostCRMDDataAsync();
    }

    // Called asynchronously when the GetCRMDData Web method returns data.
    void proxy_GetCRMDDataCompleted(object sender, GetCRMDDataCompletedEventArgs e)
    {
        // Do something with the data returned.
        string result = e.Result.ToString();
    }

    // Called asynchronously when the PostCRMDData Web method returns data.
    void proxy_PostCRMDDataCompleted(object sender, PostCRMDDataCompletedEventArgs e)
    {
        // Do something with the data returned.
        string result = e.Result.ToString();
    }
}
```

and also look at working with dynamic entities. Finally, I'll look at how to deal with the result sets that are returned to Silverlight. In the end, you'll be able to build your own Silverlight/CRM 4.0 integrations with ease.

Creating the Silverlight 4 Application

There are a number of steps to work through in order to get a Silverlight application configured to interact with CRM 4.0. Although it's possible to reference the CRM 4.0 SDK or Web service API directly from the Silverlight project, most of the interfaces and methods can't actually be called from code. In order to interact with the CRM 4.0 API, a wrapper Web service must be created. This Web service will broker the calls between the Silverlight application and the CRM 4.0 API in a format that can be handled by Silverlight. The wrapper Web service can be added directly to the SilverlightCRMDemo.Web application.

Start this process by creating a new Silverlight solution in Visual Studio 2010 called CRM40SilverlightDemo. When creating a Silverlight application in Visual Studio, two projects are always created. One is the core Silverlight application; the second is the ASP.NET application that embeds the Silverlight application into a Web page. This ASP.NET application will also be home to the Web service wrapper that will interact with the CRM 4.0 API. The Silverlight application will reference this Web service via a Service Reference.

To create the Web service wrapper, add a new Web service to the ASP.NET application and call it CrmServiceWrapper. For this example, you'll add two Web methods to the service—one to get

CRM data and one to post CRM data. Figure 1 shows what these stubbed-out methods should look like for now. Once you get the Silverlight application communicating successfully with this wrapper service, you'll update these methods to call the actual CRM 4.0 API.

Once the Web service wrapper has been added to the ASP.NET application, the easiest way to add a reference to it from the Silverlight application is to run it in debug mode and capture the URL where the debugger is running the application from (you can simply grab the URL from the browser window that pops up). Once captured, you can add a new Service Reference called *CrmServiceReference* to the Silverlight application and paste in this URL. All related configuration files and code will be automatically corrected. If you decide not to do this, you'll have to deal with cross-domain reference exceptions—and quite a bit more setup in order to successfully debug your application.

Now that the reference exists, the real coding within the Silverlight application can take place. The code consists of wiring up an event handler for each Web method and creating two methods to handle data once the calls to those Web methods have completed. The code shown in Figure 2 can be added directly to the MainPage.xaml.cs file in the Silverlight application. Running this will cause both methods to execute simultaneously.

When you've validated that your Silverlight application is running without errors and returning data from the Web service, you can turn your attention to building out the calls to the CRM 4.0 API. These calls will all be contained within the Web service wrapper GetCRMDData and PostCRMDData Web methods that have already been created.

Interacting with the CRM 4.0 API

There are two primary Web services available through CRM 4.0: the CrmService and the MetadataService. These Web services are generally available to reference from any project (but won't actually allow for much functionality when referenced from a Silverlight application). The most common and efficient way to work with the API is using the Microsoft Dynamics CRM SDK (available for download at bit.ly/6M3PW). The SDK contains numerous classes and methods, and it simplifies the communication between .NET Framework code and the CRM 4.0 Web services. In this section, you'll learn how to interact with the API using the SDK from the Web service wrapper.

The first step is to reference the appropriate CRM SDK assemblies. In the ASP.NET application that houses the Web service wrapper, add a reference to two SDK assemblies: *microsoft.crm.sdk.dll* and *microsoft.crm.sdktypeproxy.dll*. Once these have been referenced,

Figure 3 The GetCRMService Method

```
static public CrmService GetCRMService()
{
    CrmService service = new CrmService();
    CrmAuthenticationToken token =
        new Microsoft.Crm.Sdk.CrmAuthenticationToken();
    token.OrganizationName = "Contoso";
    service.Url = "http://localhost:5555/MSCRMServices/2007/crmservice.asmx";
    service.Credentials = System.Net.CredentialCache.DefaultCredentials;
    service.CrmAuthenticationTokenValue = token;
    return service;
}
```

Figure 4 Querying the System User Entity

```
[WebMethod]
public string GetCRMDData()
{
    // This will return all users in CRM in a single XML structure.
    StringBuilder xml = new StringBuilder();
    CrmService service = GetCRMService();

    QueryExpression query = new QueryExpression();
    query.EntityName = "systemuser";
    query.ColumnSet = new AllColumns();

    RetrieveMultipleRequest retrieve = new RetrieveMultipleRequest();
    retrieve.Query = query;
    retrieve.ReturnDynamicEntities = false;

    RetrieveMultipleResponse retrieved =
        (RetrieveMultipleResponse)service.Execute(retrieve);

    xml.Append("<Users>");

    for (int i = 0; i <
        retrieved.BusinessEntityCollection.BusinessEntities.Count; i++)
    {
        systemuser user =
            (systemuser)retrieved.BusinessEntityCollection.BusinessEntities[i];

        // Create a string representation to return to Silverlight app.
        xml.Append("<User>");
        xml.Append(" FirstName = '" + user.firstname + "'");
        xml.Append(" LastName = '" + user.lastname + "'");
        xml.Append(" SystemUserId = '" + user.systemuserid.ToString() + "'");
        xml.Append(" JobTitle = '" + user.jobtitle + "'");
        xml.Append(">");
    }

    xml.Append("</Users>");

    return xml.ToString();
}
```

add the appropriate directives to the top of the CrmService-Wrapper.aspx page as follows:

```
using Microsoft.Crm.Sdk;
using Microsoft.Crm.SdkTypeProxy;
using Microsoft.Crm.Sdk.Query;
```

The next step is to write code to instantiate a CRM service that will allow you to connect to a CRM 4.0 instance. This code will be used by both the GetCRMDData and PostCRMDData Web methods, so it will reside in its own method. This method (shown in **Figure 3**) requires two key fields: the organization name of your CRM 4.0 instance and the URL of the main CRM service (located at /MSCRMServices/2007/crmservice.aspx). Note that these fields are best housed in a configuration file for easy modification after compiling the code.

Querying CRM Data

You can now turn your attention to querying data from CRM 4.0. It's important to know there are two types of entities in CRM 4.0: core system entities and custom entities. The core system entities can be worked with a little more easily than custom entities. By default, all of the properties on core entities can be retrieved through strongly typed objects in C#. Custom entities are generally queried as Dynamic Entities, although an alternative allows them to be treated as strongly typed objects, also. I'll demonstrate querying data with both types of entities.

An example of querying a core system entity—the System User entity—is shown in **Figure 4**. This code replaces the Web method by the same name that was stubbed out earlier in this article. In this new

code, which queries CRM 4.0 for all system users, you'll see several important things. First, the type of object dealt with is "systemuser." All of the core entities have their own types. Second, the result being returned is a string representation of an XML document.

You'll find that the options for returning data to Silverlight are pretty limited. For example, you can't return the BusinessEntityCollection to work with, because Silverlight can't work with the API directly. Second, there are limitations with passing XML to Silverlight via a Web service. So, in the end, dealing with a simple string is likely your best option.

Querying custom entities can be a little more involved. The most common way to retrieve data is by using a Dynamic Entity to retrieve the results (an example of this kind of retrieval is shown in **Figure 5**). The challenge to this approach is in dealing with specific attributes in filters within query expressions. While everything is possible, IntelliSense can't help out much.

Although the Dynamic Entity approach may be the most common, if you want to have full visibility into the structure of your entity in Visual Studio and interact with it in the same way as a standard entity, you can create a proxy to the CrmService. In many cases, this can greatly improve the development experience and enable more flexibility in how code can be written. A proxy is nothing more than a generated C# file, based on the most current instance of the CrmService WSDL. To create a proxy class for the main CrmService service, open a Visual Studio command prompt and type the following, replacing the URL with the appropriate link to your crmservice.aspx page:

```
wsdl.exe /out:CrmSdk.cs /namespace:CRM40SilverlightDemo.WebReferences.CrmSdk
http://localhost:5555/mscrmservices/2007/crmservice.aspx?wsdl
```

This command will create a C# file called CrmSdk.cs in the directory from where you ran the wsdl.exe executable. This file should be added to your project. Once added, you can work with any custom

Figure 5 Retrieval Using a Dynamic Entity

```
public static DynamicEntity GetCRMDEntity(
    CrmService tmpService, String entityId, String entityName)
{
    DynamicEntity crmEntity = null;

    TargetRetrieveDynamic targetRetrieve = new TargetRetrieveDynamic();

    // Set the properties of the target.
    targetRetrieve.EntityName = entityName;
    targetRetrieve.EntityId = new Guid(entityId);

    // Create the request object.
    RetrieveRequest retrieve = new RetrieveRequest();

    // Set the properties of the request object.
    retrieve.Target = targetRetrieve;
    retrieve.ColumnSet = new AllColumns();

    // Retrieve as a DynamicEntity.
    retrieve.ReturnDynamicEntities = true;

    // Execute the request.
    RetrieveResponse retrieved = (RetrieveResponse)tmpService.Execute(retrieve);

    // Extract the DynamicEntity from the request.
    DynamicEntity entity = (DynamicEntity)retrieved.BusinessEntity;

    crmEntity = entity;

    return crmEntity;
}
```


Figure 6 Posting CRM 4.0 Data

```
[WebMethod]
public string PostCRMDData(string userId, string nickname)
{
    CrmService service = GetCRMService();

    Key kid = new Key();
    kid.Value = new Guid(userId);
    systemuser entity = new systemuser();
    entity.systemuserid = kid;

    entity.nickname = nickname;

    service.Update(entity);

    return "success";
}
```

entity in the exact same manner as core system entities. If the entity changes, simply update your proxy C# file and the new attributes (or other modifications) will be available. For the purposes of the current exercise, the proxy C# file won't be used.

Updating CRM 4.0 Data

Having looked at how to retrieve data from CRM 4.0, you can now work through posting data. The code shown in **Figure 6** shows how to update a system user record. It requires that two properties be passed in: the unique identifier of the CRM 4.0 record and the nickname. In order to pass in these strings, one line of code must be modified in the MainPage.xaml.cs, shown here:

```
proxyPost.PostCRMDDataAsync("f04b02d9-ad5f-e011-a513-000c29330bd5", "My Nickname");
```

Notice that the ID is hardcoded in the call to the PostCRMDData method. You'll want to come up with a mechanism to grab the ID dynamically.

Processing the Results in Silverlight

At this point, the solution should be retrieving and posting data to and from CRM 4.0. However, nothing is being done with the string results that are returned to Silverlight. The GetCRMDData method is returning

a string of data that contains an XML document with all of the user records, but what can be done with that? Depending on the control, you may be able to bind to XML directly, or you may want to parse through the XML that's returned and deal with individual data elements.

An example of looping through the results returned can be seen in **Figure 7**. This code shows how to load the string into an XML document and loop through the data. In working with XML documents in Silverlight, the most versatile functionality comes from the XDocument class. This can be accessed by adding a reference to System.Xml.Linq in your Silverlight project.

Endless Possibilities

There are endless possibilities when integrating the two technologies. Some of the immediate next steps you might want to look at are approaches to exception handling (many Silverlight controls hide exceptions, so you'll need to deal with this on a case-by-case basis) and integrating with various controls. Regardless of the direction you take, you've now got everything you need in order to build Silverlight solutions that read and write data with CRM 4.0. ■

MARK BECKNER is the founder of Inotek Consulting Group LLC. He works across the Microsoft stack, including BizTalk, SharePoint, Dynamics CRM and general .NET Framework development. He can be reached at mbeckner@inotekgroup.com.

THANKS to the following technical expert for reviewing this article: Scott Jones


Figure 7 Working with XDocuments

```
void proxy_GetCRMDDataCompleted(object sender, GetCRMDDataCompletedEventArgs e)
{
    XDocument xDoc = XDocument.Parse(e.Result);

    string firstName;
    string lastName;
    string ID;
    string title;

    // Loop through the results.
    foreach (XElement element in xDoc.Descendants("User"))
    {
        firstName = GetAttributeValue(element, "FirstName");
        lastName = GetAttributeValue(element, "LastName");
        ID = GetAttributeValue(element, "SystemUserId");
        title = GetAttributeValue(element, "JobTitle");
    }
}

private string GetAttributeValue(XElement element, string strAttributeName)
{
    if (element.Attribute(strAttributeName) != null)
    {
        return element.Attribute(strAttributeName).Value;
    }
    return string.Empty;
}
```



GoDiagram

Add powerful diagramming capabilities to your applications in less time than you ever imagined with GoDiagram components.

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.godiagram.com

Building Information Architecture in SharePoint 2010

Shahram Khosravi, Ph.D.

The release of Microsoft SharePoint 2010 saw new Enterprise Content Management (ECM) capabilities added to the collaboration software. This article shows you how to take advantage of these new features to build flexible, extensible and maintainable information architecture for the following two types of portals:

- Internet/intranet/extranet-facing publishing portals
- Knowledge-management portals

I'll walk you through the design and implementation of several custom SharePoint components that can be used to implement the information architecture for these portals.

This article discusses:

- Building information architecture for Internet/intranet/extranet-facing publishing portals
- Implementing a custom data source control
- Using a ListSiteMapPath control to solve a problem
- Building information architecture for knowledge-management portals
- The knowledge-management portal homepage
- Building a wiki page

Technologies discussed:

SharePoint 2010

Code download available at:

code.msdn.microsoft.com/mag201110SPECM

Building Information Architecture for Internet/Intranet/Extranet-Facing Publishing Portals

Traditionally, Internet/intranet/extranet-facing publishing portals follow a strictly guided navigation model where GUI elements such as menus are used to guide the users to the content they're looking for. Such portals normally have a top and a left navigation menu.

The top navigation menu is usually an AspMenu that allows users to make selections from the first and second tiers of the portal taxonomy. The left navigation menu is usually an AspMenu that allows users to make selections from the third and fourth (or possibly fifth) tiers of the portal taxonomy.

The entire content of the portal is divided into a set of categories that form the first tier. I'll use the following naming convention for categories: Term i where i takes on an integer value. For example, the first and second categories are named Term1 and Term2, respectively.

The content in each category is then divided into a set of subcategories which form the second tier. I'll use the following naming convention for subcategories: Term ij where i and j take on integer values. For example, the first subcategory of the first category is named Term11.

This categorization of the content continues until we reach the desired granularity. **Figure 1** shows this portal taxonomy.

Such portal taxonomy is normally implemented in SharePoint 2007 through establishment of the appropriate site structure. I'll start by creating a separate site for each category. I'll then create

separate subsites under each category site for the respective subcategories, and so on. **Figure 2** shows the site structure of the portal.

Basically, each category, subcategory, sub-subcategory and so on has its own dedicated site. This dedicated site contains a Pages document library that contains the pages for that site. If I were to use social networking language, I could say all pages in the Pages document library of a site are implicitly tagged with the category, subcategory or sub-subcategory that the site represents. In a way, I implicitly tag a bunch of pages with a category, subcategory or sub-subcategory by creating them in a site that represents that category, subcategory or sub-subcategory. This way, I create a new implicit tag (or term) by creating a new site.

The top navigation AspMenu displays these categories (first tier) and their subcategories (second tier). The user selects a category or subcategory from this AspMenu to navigate to the page that's implicitly tagged with that category or subcategory.

When the user selects a subcategory from the top navigation menu, the left navigation AspMenu (Quick Launch) displays the sub-subcategories associated with that subcategory. The user selects a sub-subcategory from this AspMenu to navigate to the page that's implicitly tagged with that sub-subcategory. The Pages document library of a given site may contain multiple pages, which means that multiple pages may be implicitly tagged with the same category, subcategory or sub-subcategory. The Quick Launch displays the links to all pages that are implicitly tagged with the same sub-subcategory.

Comparison of **Figure 1** and **Figure 2** clearly shows that the portal taxonomy or information architecture directly reflects the portal site structure. This makes the information architecture inflexible and introduces the following problems:

- Creating a new implicit tag requires site administration permission to create a new site.
- Retagging a page requires physically moving the page from one site to another.
- Reorganizing the information architecture requires physically moving and deleting sites and pages.
- Content authors aren't able to share the same page across multiple categories, subcategories, sub-subcategories and so on. They have to create a separate page for each category, subcategory and so on because each is a separate site. The only out-of-the-box option is to copy the page to the respective sites. This solution introduces two usability problems. First, the content author has to copy the page to many different places. Second, each time the content author changes the content in one page, he has to go back to all those copies in all those sites to make the same updates or copy the page again. This is error-prone.

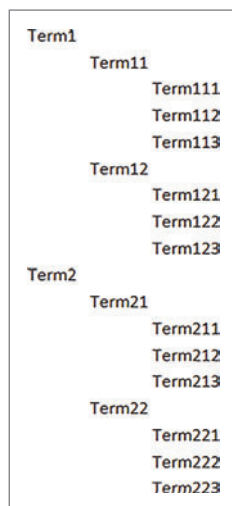


Figure 1 Portal Taxonomy

- Because changing taxonomy requires structural changes in the site structure, which involves a lot of time and effort, taxonomy is very rigid. Taxonomy is tightly coupled with the way information is actually stored in the site collection.

Enter SharePoint 2010 ECM. You can now implement your portal taxonomy in a managed metadata service application where you manage it centrally. Such implementation no longer depends on your portal site structure:

- You don't need to provision new sites just to create new implicit tags. You simply add new terms to the desired spots in the portal taxonomy in the term store. This way, you can now keep all your pages in the same Pages document library in the same site because scalability of document libraries is no longer an issue in SharePoint 2010. This article assumes that all pages are maintained in a single Pages document library in a single site.
- You can retag a page with a new term without having to physically move the page.
- You can reorganize your taxonomy by simply reorganizing your taxonomy in the term store without having to physically move or delete sites and pages.
- You can share the same page across multiple categories, subcategories and so on by simply tagging the page with multiple terms—where each term represents a category, subcategory and so on—without having to physically copy the page across sites.

I'll implement the portal taxonomy as a single term set, as shown in **Figure 3**.

However, implementing the portal taxonomy in a term store instead of the portal site structure introduces two challenges discussed in the next sections.

The TaxonomyDataSource Control

First, the v4.master master page comes with an AspMenu that displays the top navigation menu. This control is bound to a site map data source control that uses a provider to retrieve the site map data from the portal site structure. This isn't going to work in this case because I want the site map data to come from the term store, not the portal's physical site structure.

One option is to implement a custom site map provider that retrieves the appropriate terms from the term store. Another option is to implement a custom data source control. I'll use the latter because ASP.NET comes with a powerful data source control named XmlDataSource that I can easily extend to achieve my goal. I'll name this custom data source control TaxonomyDataSource.

TaxonomyDataSource exposes a Boolean property named IsGlobal. I'll include two instances of Taxonomy-

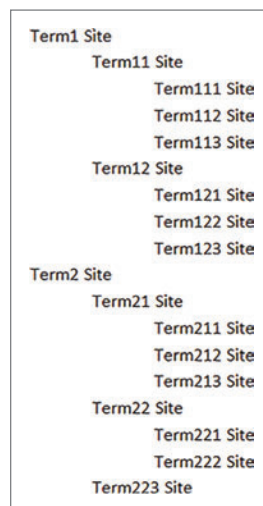


Figure 2 The Site Structure of the Portal

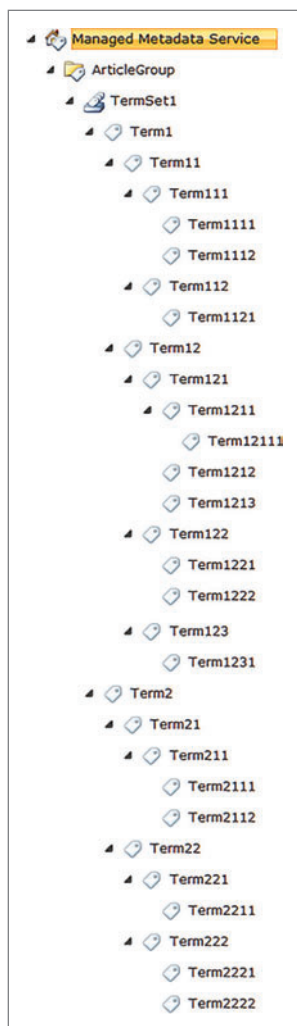


Figure 3 Implementation of Portal Taxonomy in the Managed Metadata Service Application

term store. The main objective of the Data property is to create the XML document that XmlDataSource uses to create the nodes that it passes to AspMenu. This property first uses the SharePoint 2010 managed metadata API to access the term set that contains the portal taxonomy, as shown here:

```
TaxonomySession taxonomySession = new TaxonomySession(SPContext.Current.Site);
TermStore termStore = taxonomySession.TermStores[this.TermStore];
Group group = termStore.Groups[this.Group];
TermSet termSet = group.TermSets[this.TermSet];
```

The property uses all terms in the term set if IsGlobal is true:
termsToReturn = termSet.Terms;

If IsGlobal is false, the property first accesses the GUID of the current term, which is exposed through the CurrentTermId query string parameter. Then it uses the child and grandchild terms of the current term:

```
Term currentTerm = taxonomySession.GetTerm(
    new Guid(this.Page.Request.QueryString["CurrentTermId"]));
termsToReturn = currentTerm.Terms;
```

The Data property then starts creating the XML document. This document has a document element named <Terms>, which contains

DataSource in the master page. One instance will be bound to the AspMenu that renders the top navigation menu. The IsGlobal property of this instance will be set to true to retrieve all terms of the term set that contains the portal taxonomy. This enables the top navigation menu to display all categories and their subcategories. Keep in mind that categories and their respective subcategories are nothing but the child and grandchild terms of this term set.

The second instance will be bound to the AspMenu that renders the left navigation menu (Quick Launch). The IsGlobal property of this instance will be set to false to retrieve only the subterms of the current term. Let me elaborate on what I mean by "current term."

Recall that when the user selects a subcategory from the top navigation menu, the left navigation menu must display the list of sub-subcategories associated with that subcategory. The current term in this context is the subcategory. The second instance basically returns the current term's child and grandchild terms, which are nothing but the sub-subcategories and sub-sub-subcategories.

TaxonomyDataSource overrides the Data property to retrieve the portal taxonomy from the

a hierarchy of <Term> elements. TaxonomyDataSource creates a separate <Term> element to represent each term that it pulls from the term store. Following is an example of such an XML document:

```
<Terms>
  <Term Name="Term1" URL="PageUrl?CurrentTermId=">
    <Term Name="Term11" URL="PageUrl?CurrentTermId=">
      <Term Name="Term111" URL="PageUrl?CurrentTermId="/>
    </Term>
  </Term>
  <Term Name="Term2" URL="PageUrl?CurrentTermId=">
    <Term Name="Term21" URL="PageUrl?CurrentTermId="/>
  </Term>
</Terms>
```

Note that the <Term> element has two attributes named Name and URL. The Name attribute is set to the name of the term and the URL is set to the URL of the page that's tagged with that term. This URL includes a query string parameter named CurrentTermId that contains the GUID of the current term.

Implementing the portal taxonomy in a term store instead of the portal site structure introduces two challenges.

The Data property iterates through the retrieved terms and invokes the GetXmlFragment method for each enumerated term. The main goal of this method is to build the <Term> element that represents the enumerated term and this <Term> element's <Term> subelements that represent the child and grandchild terms of the enumerated term:

```
foreach (Term term in termsToReturn)
{
    GetXmlFragment(publishingPageCollection, term, ref xml);
}
```

Note that TaxonomyDataSource uses the SharePoint publishing API to access the collection that contains the publishing pages.

Next, I'll discuss the implementation of GetXmlFragment. However, first you need to understand the significance of the Tags site

Figure 4 The First Part of GetXmlFragment

```
foreach (PublishingPage publishingPage in publishingPageCollection)
{
    TaxonomyFieldValueCollection values =
        publishingPage.ListItem["Tags"] as TaxonomyFieldValueCollection;

    foreach (TaxonomyFieldValue value in values)
    {
        if (value != null && value.TermGuid == term.Id.ToString())
        {
            url = publishingPage.Uri.AbsoluteUri;
            xml += "<Term Name='" + term.Name + "' URL='" + url +
                "?CurrentTermId=" + term.Id.ToString() + "'>";
            closeTerm = true;
            defaultPublishingPage = publishingPage;
            break;
        }
    }

    if (closeTerm)
        break;
}
```

We didn't invent the Internet...

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



TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
AS2, EDI/X12, NAESB, OFTP ...
- **Credit Card Processing**
Authorize.Net, TSYS, FDMS ...
- **Shipping & Tracking**
FedEx, UPS, USPS ...
- **Accounting & Banking**
QuickBooks, OFX ...
- **Internet Business**
Amazon, eBay, PayPal ...
- **Internet Protocols**
FTP, SMTP, IMAP, POP, WebDav ...
- **Secure Connectivity**
SSH, SFTP, SSL, Certificates ...
- **Secure Email**
S/MIME, OpenPGP ...
- **Network Management**
SNMP, MIB, LDAP, Monitoring ...
- **Compression & Encryption**
Zip, Gzip, Jar, AES ...



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.

connectivity
powered by 

To learn more please visit our website →

www.nsoftware.com

Figure 5 The Third Part of GetXmlFragment

```
if (!this.IsGlobal)
{
    foreach (PublishingPage publishingPage in publishingPageCollection)
    {
        if (publishingPage == defaultPublishingPage)
            continue;

        TaxonomyFieldValueCollection values =
            publishingPage.ListItem["Tags"] as TaxonomyFieldValueCollection;

        foreach (TaxonomyFieldValue value in values)
        {
            if (value != null && value.TermGuid == term.Id.ToString())
            {
                url = publishingPage.Uri.AbsoluteUri;
                xml += "<Term Name='" + publishingPage.Title + "' URL='" +
                    url + "?CurrentTermId=" + term.Id.ToString() + "'/>";
                break;
            }
        }
    }
}
```

column, which is of type Managed Metadata. You'll need to create this site column and bind it to the term set that contains the portal taxonomy and add the site column to the associated content type of the publishing page layout. The Tags column allows the content author to tag publishing pages with terms from the portal taxonomy.

GetXmlFragment consists of three parts. As **Figure 4** shows, the first part searches through the pages in the Pages document library for the first page that's tagged with the specified term and renders a <Term> element to represent the term.

The URL attribute of this <Term> element is set to the URL of this page and the CurrentTermId query string parameter is set to the GUID of this term. This page acts as the default page for the specified term. This basically simulates the default page of a site if I were to create a site to represent the specified term.

The code in **Figure 4** basically generates the following XML fragment:

```
<Term Name='TermName' URL='DefaultPageURL?CurrentTermId=GUID'
```

Note that the <Term> element isn't closed yet because I still need to iterate through the child and grandchild terms of this term and render <Term> subelements within this <Term> element for each enumerated child and grandchild term. This is exactly what the second part of GetXmlFragment does:

```
foreach (Term cterm in term.Terms)
{
    GetXmlFragment(publishingPageCollection, cterm, ref xml);
}
```

GetXmlFragment finally closes the parent <Term> element. Next, GetXmlFragment renders <Term> elements for the rest of the pages in the Pages document library that are tagged with the same term. These <Term> elements are treated as the siblings of the <Term> element that represents the default page. This allows Quick Launch to render the links to all pages that are tagged with the current term. This basically simulates the non-default pages of a site if I were to represent the term with a site, as shown in **Figure 5**.

Figure 6 shows an example of a page based on the master page that uses TaxonomyDataSource.

Thanks to TaxonomyDataSource, the top and left navigation menus display different tiers of the portal taxonomy, which is maintained centrally in the term store.

The ListSiteMapPath Control

As mentioned, implementing the portal taxonomy in a term store instead of the portal site structure introduces two challenges. In the previous section I discussed the first challenge and provided a solution. This section discusses the second challenge and provides a solution to address it.

v4.master contains a control named PopoutMenu that renders a control named ListSiteMapPath. ListSiteMapPath renders the breadcrumb that shows up when the user clicks PopoutMenu, which is shown as an icon on the page.

ListSiteMapPath inherits from SiteMapPath, which is a traditional control for rendering the breadcrumb. Also, default.master uses SiteMapPath, whereas v4.master uses ListSiteMapPath.

ListSiteMapPath exposes a property named SiteMapProviders that takes a comma-separated list of site map provider names. As such, ListSiteMapPath can work with more than one provider. ListSiteMapPath iterates through these providers and takes the following steps for each enumerated provider. It first invokes the FindSiteMapNode method of the provider, passing in HttpContext to return the current site map node. It then recursively invokes the GetParentNode method of the provider to access all ancestor site map nodes of the current site map node all the way up to the root site map node. Finally, it renders an item in the breadcrumb for each ancestor site map node starting from the root site map node all the way down to the current site map node.

As you can see, ListSiteMapPath renders a complete hierarchy of items for each enumerated site map provider. This way, each site map provider contributes to the breadcrumb. This enables ListSiteMapPath to render site map nodes from different types of sources. In my case, I want to have ListSiteMapPath render site map nodes from the portal taxonomy maintained in the term store. I'll implement a custom site map provider named TaxonomySiteMapProvider to achieve this.

TaxonomySiteMapProvider overrides FindSiteMapNode, where it uses the specified context to access the respective SPLItem. This context is basically the current context. The method then accesses the URL, ID and title of this list item and creates a site map node to represent it. Keep in mind that this list item in publishing sites represents a publishing page.

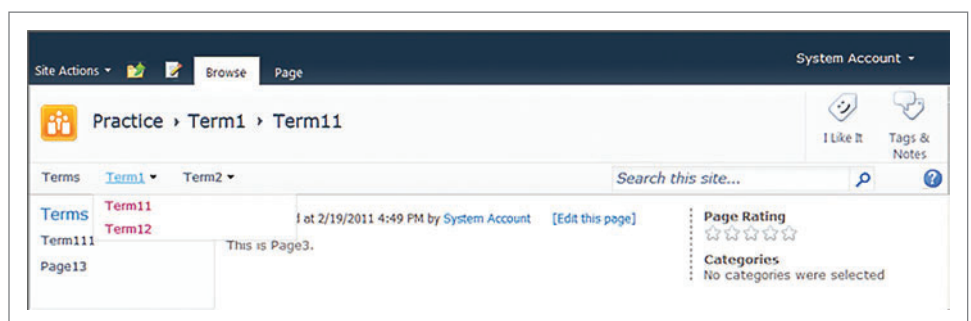


Figure 6 An Example Page Based on a Master Page that Uses TaxonomyDataSource

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



Figure 7 Forming the Parent Site Map Node

```
Term parentTerm = term.Parent;
if (parentTerm != null)
{
    int[] wssIds = TaxonomyField.GetWssIdsOfTerm(context.Site, parentTerm.TermStore.Id,
        parentTerm.TermSet.Id, parentTerm.Id, false, 500);
    query = new SPQuery(list.DefaultView);
    query.Query =
        "<Where><In><FieldRef LookupId='True' Name='Tags' /><Values>";
    foreach (int wssId in wssIds)
    {
        query.Query += ("<Value Type='Integer'>" + wssId.ToString() + "</Value>");
    }
    query.Query += "</Values></In></Where>";
    listItems = list.GetItems(query);
    listItem = listItems[0];

    string url = listItem.Url;
    string key = listItem.ID.ToString();
    string title = parentTerm.Name;

    return new SiteMapNode(this, key, url, title);
}
```

ListSiteMapPath invokes this method to access the current site map node:

```
SPContext spContext = SPContext.GetContext(context);
SPLISTItem listItem = spContext.ListItem;
string url = listItem.Url;
string key = listItem.ID.ToString();
string title = listItem.Title;
return new SiteMapNode(this, key, url, title);
```

ListSiteMapPath then invokes GetParentNode, passing in the current site map node to return the site map node that represents the parent of the current node. Keep in mind that the current site map node represents the current publishing page and the parent site map node represents the parent of this publishing page. The parent of a publishing page in this case is the publishing page that's tagged with the parent term of the term with which the current publishing page is tagged.

Social networking portals promote the free-flow approach to navigation as opposed to the linear guided approach used in traditional portals.

The TaxonomyDataSource override of GetParentNode first accesses the current SharePoint list, which is nothing but the Pages document library that contains the publishing pages. It then creates a Collaborative Application Markup Language (CAML) query to search this list for the current publishing page so it can access the term with which the current page is tagged. This time around, I'm not using the publishing API to access this publishing page. Using a CAML query provides better performance, especially if the Pages document library contains tens of thousands of pages.

The following code section of GetParentNode basically returns the publishing page that the specified node represents, and then retrieves the value of the Tags field, which is the term with which the page is tagged:

```
SPQuery query = new SPQuery();

query.ViewFields = "<FieldRef Name='Tags' />";
query.Query = "<Where><Eq><FieldRef Name='ID' /><Value Type='Integer'>" +
    node.Key + "</Value></Eq></Where>";
SPLISTItemCollection listItems = list.GetItems(query);
SPLISTItem listItem = listItems[0];

TaxonomyFieldValueCollection values =
    listItem["Tags"] as TaxonomyFieldValueCollection;
TaxonomyFieldValue value = values[0];

Guid termGuid = new Guid(value.TermGuid);
TaxonomySession taxonomySession = new TaxonomySession(context.Site);
Term term = taxonomySession.GetTerm(termGuid);
```

Next, GetParentNode accesses the parent term of the current term and searches the Pages document library for the publishing page that's tagged with this parent term. It finally accesses the URL and ID of this publishing page and uses these two pieces of information, plus the name of the parent term, to form the parent site map node, as shown in Figure 7.

The code in Figure 7 uses the GetWssIdsOfTerm static method of the TaxonomyField class. This method returns the Windows SharePoint Services (WSS) IDs of the specified term from the taxonomy hidden list, which is a SharePoint list at the site-collection level where SharePoint caches terms used in the site collection. Every time you tag an item with a new term, SharePoint automatically adds an entry to this hidden list for that term. The same term may have multiple entries in this taxonomy hidden list if, for example, the term is reused.

GetWssIdsOfTerm returns the IDs of all list items that represent the specified term in the taxonomy hidden list. You have to use these IDs in the CAML queries that filter based on managed metadata fields, which is the Tags field in this case.

Figure 8 shows the ListSiteMapPath that's bound to the TaxonomySiteMapProvider. Note that the control renders the entire hierarchy to which the current page belongs.

Building Information Architecture for Knowledge-Management Portals

As discussed, a traditional guided-navigation approach takes users through a set of menu options to help them find the content they're looking for. This is quite different from social networking portals, where approaches such as the following are used to help users find content:

- Search engine: Users perform search queries against search indices to find content.
- Metadata filtering: Users use metadata filtering to filter search results.
- Wiki-style links: Pages are linked together through wiki-style links. Users use these links to navigate between pages.
- Rollup Web Parts: Pages contain Rollup Web Parts that roll up content from various places. Users use the links in these Web Parts to navigate between pages.

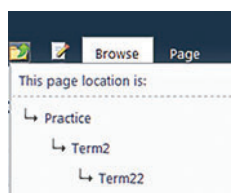


Figure 8 The ListSiteMapPath that Is Bound to the TaxonomySiteMapProvider

ROCK STARS WANTED!

The **Visual Studio Gallery** is home to thousands of Visual Studio extensions built by rock star developers like you. Tools, controls and templates from the Visual Studio Gallery show up within the IDE and make it easy for others to create great products. Share the tools you build with *millions of developers*.

...> **SHARE WHAT YOU BUILD** | visualstudiogallery.com



Microsoft®

Visual Studio® Gallery



Extension developers, take it to the next level. The Visual Studio Industry Partner (VSIP) program helps you build, market and sell products that integrate with Visual Studio.

...> **BECOME A PARTNER** | msdn.com/vsip

Figure 9 The New Version of TaxonomyDataSource

```
TaxonomyFieldValueCollection values =  
    SPContext.Current.ListItem["Tags"] as TaxonomyFieldValueCollection;  
Term currentTerm = null;  
foreach (TaxonomyFieldValue value in values)  
{  
    currentTerm = taxonomySession.GetTerm(new Guid(value.TermGuid));  
    GetXmlFragment(publishingPageCollection, currentTerm, ref xml);  
    termsToReturn = currentTerm.Terms;  
  
    if (termsToReturn != null)  
    {  
        foreach (Term term in termsToReturn)  
        {  
            GetXmlFragment(publishingPageCollection, term, ref xml);  
        }  
    }  
}
```

Social networking portals promote the free-flow approach to navigation as opposed to the linear guided approach used in traditional portals. Thus, users can arrive at a page through different means. In the following sections, I'll discuss the design and implementation of a knowledge-management portal that takes advantage of the social networking navigation model. Sean Squires and Lincoln DeMaris presented this design during a session at the Microsoft SharePoint Conference 2009. I'll cover some aspects of this design and also present my own customizations and enhancements to it. The next section discusses the homepage of this knowledge-management portal.

The Homepage

The main goal of the homepage is to provide the following two capabilities: search box and Rollup Web Parts. The search box enables users to search for content. This is the main component of the homepage. The Rollup Web Parts roll up content from various places throughout the enterprise.

This content could be anything that's of some interest to users. For example, it may be beneficial to users to show them the most-viewed and most-searched content in the enterprise. SharePoint 2010 comes with an out-of-the-box Web Analytics Web Part that you can configure to display such content.

A knowledge-management portal normally stores documents in a central repository. SharePoint comes with an out-of-the-box

site template named Document Center. You can create a site from this site template to store your documents. You can then add a content query Web Part to the homepage to roll up the most recently uploaded documents.

When the user uses the search box shown in the homepage to perform a search query, she's taken to the search results page. This page is the standard SharePoint out-of-the-box search results page that comes with a refinement panel. This panel contains categories that the user can use to filter search results. One of these categories is result types.

The user selects the result type associated with Web pages to view Web pages only. The refinement panel also presents the terms with which these Web pages are tagged. These are the terms from the portal taxonomy. The user uses these terms to further filter these pages to arrive at the desired page. The user then clicks on the search result to navigate to the page, which is basically an enterprise wiki page.

A knowledge-management
portal normally stores documents
in a central repository.

Structure of an Enterprise Wiki Page

All pages in my knowledge-management portal are created from the same customized version of the enterprise wiki page layout.

To customize the page layout, I start by replacing the Wiki Categories field control with the Tags field control. I'll also place a Tags field control inside an EditModePanel so that it only shows up when the page is in Edit mode. The reason I don't want the Tags field control to show up in the Display mode is because I'll use the left navigation menu to render the terms with which the current page is tagged instead of the Tags field control. As you'll see shortly, using AspMenu will allow me to not only render the current terms, but also the current terms' subterms, sub-subterms and so on.

I also need to make changes in the implementation of TaxonomyDataSource so the control doesn't use the query string to get the current term. Instead, it should retrieve the current terms from the Tags field of the current page. This is because I'm no longer taking the user through the top navigation menu, which is how it's done in a guided-navigation model used in traditional portals. The user can access the current page from anywhere. For example, there could be a wiki link on some page that takes the user to the current page. As such, there's no query string parameter that would tell you about the current term. This also

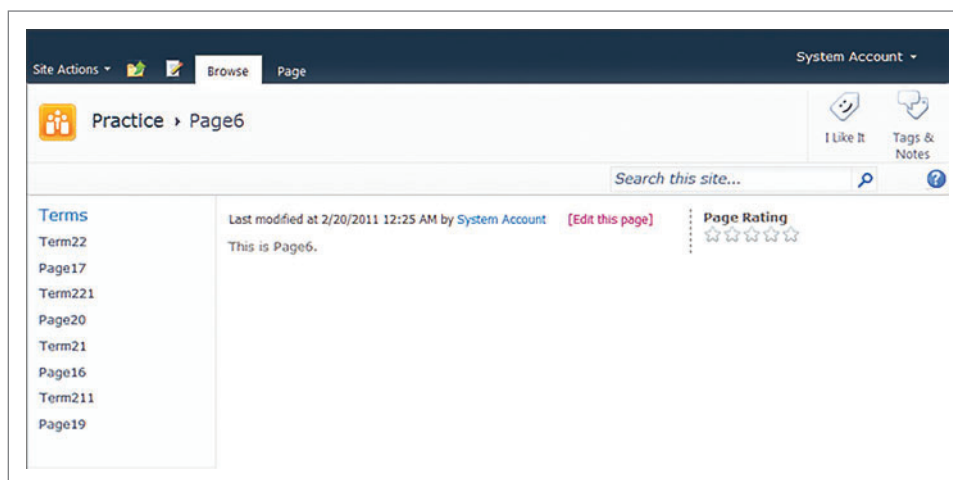


Figure 10 A Page Based on a Customized Enterprise Wiki Layout

ALM Summit

application lifecycle management
for the microsoft platform

sponsored by Microsoft

Register at www.alm-summit.com

follow us on



November 14-18, 2011 - Microsoft Conference Center, Redmond

Conference Workshop Options:

Pre-conference (Monday)

Implementing Scrum - Richard Hundhausen

Managing your application lifecycle with TFS - Anthony Borton

Post-conference (Friday)

Enterprise Management of the Software Process - Steven Borg

Visual Studio 2010 Lab Management - Brian Randell

keynote speakers



Tim Lister
Author



Jason Zander
Corporate Vice President
Microsoft



Scott Guthrie
Corporate Vice President
Microsoft



Dave West
Senior Analyst
Forrester Research



Mark Russinovich
Technical Fellow
Microsoft



Brian Harry
Technical Fellow
Microsoft

track chairs



Sam Guckenheimer
ALM Leadership Track



Jim Newkirk
Agile Developer Track

sponsors



diamond

platinum

means that the current term may no longer be a single term, because the current page may have been tagged with multiple terms.

The new version of TaxonomyDataSource first accesses all terms with which the current page is tagged. It then iterates through these terms and creates XML fragments for these terms and their descendant terms, as shown in **Figure 9**.

Figure 10 shows a page based on my customized version of an enterprise wiki page layout.

Note that Quick Launch displays the current terms, the subterms of the current terms and the pages that are tagged with the current terms and subterms of the current terms. Also note that the page doesn't display the Tags field control in the Display mode.

Figure 11 shows the page in the Edit mode where the Tags field is shown, allowing the content author to tag or retag the page.

Thanks to TaxonomyDataSource, the left navigation menu exposes links to all pages that are related to the current page. It exposes links not only to the pages that are tagged with the same terms as the current page, but also to pages that are tagged with the descendant terms of these terms. You can enhance TaxonomyDataSource to give the content author the option of specifying whether to display only the child terms of the terms with which the current page is tagged, the child terms and the grandchild terms, or the child terms, grandchild terms and the grand-grandchild terms, and so on. You can also give the author the option to display the parent terms of the terms with which the current page is tagged, the parent terms and the grandparent terms, and so on.

Links on the Quick Launch are just
one way to make connections
between the pages of your portal.

Thus, TaxonomyDataSource becomes a powerful tool that ensures when a user visits a page, the page presents the user with links to all related pages throughout the enterprise. As more pages are added and tagged with the same terms with which the current page is tagged or with the descendant terms of the same terms, TaxonomyDataSource automatically picks them up, so Quick Launch is automatically updated to include links to these new pages. TaxonomyDataSource makes it possible to use the portal taxonomy as the common thread that chains together all pages and documents throughout your enterprise.

Links on the Quick Launch are just one way to make connections between the pages of your portal. Another way is to use wiki-style page linking. Yet another means is to use Rollup Web Parts that bring

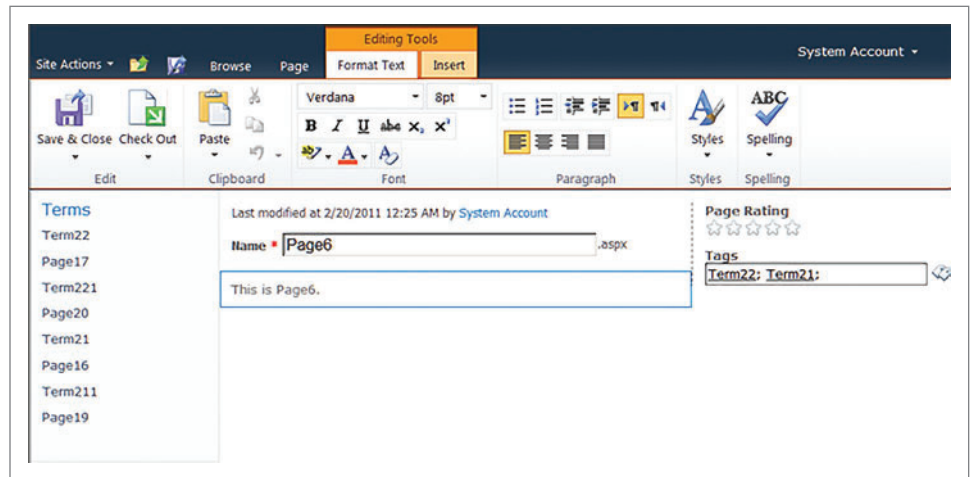


Figure 11 A Page in Edit Mode, Allowing Tagging or Retagging

in related content from throughout the enterprise. For example, you can configure a Content Query Web Part to pull documents from your document center and filter them based on the terms with which the current page is tagged. As users upload new documents to the document center and tag them with the same terms with which the current page is tagged, this Content Query Web Part automatically updates to show these new documents as well.

As the content author adds a new tag to the page, edits the existing tag or removes an existing tag, the content of all Rollup Web Parts on the page and the content of Quick Launch automatically changes. This is an example of *contextual social networking*, where the page displays content based on the taxonomy.

TaxonomyDataSource, wiki-style linking and Rollup Web Parts turn a page into a forefront for pulling related content from throughout the enterprise to one location, filtered based on the terms with which the page is tagged. Pages are no longer explicitly structured through physical structure of the portal. Instead, they're implicitly structured via the relationships they have through taxonomy, wiki-style linking and Rollup Web Parts.

Flexible and Customized Information Architecture

Wrapping up, the new SharePoint 2010 ECM capabilities help you build and implement flexible information architecture for Internet/intranet/extranet-facing publishing and knowledge-management portals. This flexibility is becoming more important as Web page and site design evolve to accommodate new ways of using the Internet, as exemplified by the explosion in social networking sites. While I presented the design and implementation of several custom SharePoint components that can be used to implement information architecture, there's much more that can be done, and I urge you to explore the new possibilities. ■

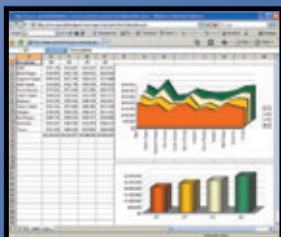
SHAHRAM KHOSRAVI specializes in and has extensive industry experience with SharePoint architecture, design and development. He's the author of the following books: "Expert WSS 3.0 and MOSS 2007 Programming" (Wrox, 2008), "Professional SharePoint 2007 Workflow Programming" (Wrox, 2008), "ASP.NET AJAX Programmer's Reference" (Wrox, 2007), "ASP.NET 2.0 Server Control and Component Development" (Wrox, 2006) and "IIS 7 and ASP.NET Integrated Programming" (Wrox, 2007).

Microsoft Chose SpreadsheetGear...

"After carefully evaluating SpreadsheetGear, Excel Services, and other 3rd party options, we ultimately chose SpreadsheetGear for .NET because it is the best fit for MSN Money."

Chris Donohue, MSN Money Program Manager

SpreadsheetGear 2010



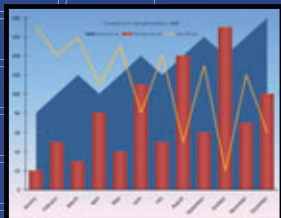
ASP.NET Excel Reporting

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.



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



Graph Structures and Maximum Clique

In this month's column, I present the design, a C# language implementation and testing techniques for a graph data structure that can be used to solve the maximum clique problem. The graph code can also be used for many other problems, as I'll explain.

So, just what is the maximum clique problem and why might it be relevant to you? A clique is a subset of a graph where every node is connected to every other node. Take a look at the graph representation in **Figure 1**. Nodes 2, 4 and 5 form a clique of size three. The maximum clique problem is to find the clique with the largest size in a graph. The maximum clique for the graph in **Figure 1** is the node set { 0, 1, 3, 4 }, which has size four.

The maximum clique problem is encountered in a wide range of applications, including social network communication analysis, computer network analysis, computer vision and many others. For graphs of even moderate size, it turns out that the maximum clique problem is one of the most challenging and interesting problems in computer science. The techniques used to solve the maximum clique problem—which include tabu search, greedy search, plateau search, real-time parameter adaptation and dynamic solution history—can be used in many other problem scenarios. In short, code that solves the maximum clique problem can be directly useful to you, and the advanced techniques employed in the algorithm can be helpful for solving other difficult programming problems.

A complete solution to the maximum clique problem is too long to present and explain in one article, so I'll present the solution over several articles. The first step for solving the maximum clique problem is to design, implement and test a data structure that can efficiently store the graph under analysis in memory. The console application in **Figure 2** shows you where I'm headed in this column.

With some WriteLine statements removed, the code that produced the run shown in **Figure 2** is:

```
string graphFile = "..\\..\\DimacsGraph.clq";
MyGraph.ValidateGraphFile(graphFile, "DIMACS");

MyGraph graph = new MyGraph(graphFile, "DIMACS");

graph.ValidateGraph();
Console.WriteLine(graph.ToString());

Console.WriteLine("\nAre nodes 5 and 8 adjacent? " +
    graph.AreAdjacent(5,8));
Console.WriteLine("Number neighbors of node 4 = " +
    graph.NumberNeighbors(4));
```

The data for the **Figure 1** graph is stored in an external text file named `DimacsClique.clq`, which uses a standard format called DIMACS. I'll explain the DIMACS file format shortly. My demo program begins by validating the source file, then instantiates a graph data structure using the data file. After the graph has been instantiated, I validate the internal representation and display it in a human-friendly image. As you'll see, an efficient internal representation of a graph is critically important for the maximum clique problem. The demo program finishes by calling a method that determines if two nodes are adjacent, nodes 5 and 8 in this case, and by calling a method that returns the number of neighbors a node has, for node 4 in this case.

For the maximum clique problem, representing a graph using a bit matrix provides an excellent combination of space and performance efficiencies.

I'll walk you through the code that generated the **Figure 2** output line by line. The complete source code for the demo program is available at code.msdn.microsoft.com/mag201110TestRun. The code is written in C#, but you should be able to follow me if you have intermediate-level programming skills in any modern high-level language. The graph code presented here lays the foundation for solving the maximum clique problem in upcoming articles and should be a useful addition to your developer, tester and software project management toolkits.

A Bit Matrix

There are several common ways to represent an unweighted (graph edges aren't assigned priorities of some sort), undirected (edges don't have a direction from one node to another) graph in memory. For the maximum clique problem, representing a graph using a bit matrix provides an excellent combination of space and performance efficiencies. **Figure 3** shows a bit matrix that corresponds to the example graph. Even though we're dealing with an undirected graph, it's common to call the vertical indices the from-nodes

Code download available at code.msdn.microsoft.com/mag201110TestRun.

and the horizontal indices the to-nodes. A value of 1 means there's an edge between the corresponding nodes; a 0 value indicates no edge between nodes. Notice that the matrix is symmetric and that we assume nodes aren't adjacent to themselves.

The primary advantage of a bit matrix over alternative designs is that it allows fast adjacency lookups, which often dominate the runtime of many graph algorithms, including the maximum clique problem. If implemented crudely, the primary disadvantage of a bit matrix is memory usage. For example, if the 9x9 matrix in Figure 3 was implemented as a two-dimensional array of 4 byte integers or Booleans, the matrix would require $9 * 9 * 4 = 324$ bytes. But because each value in a bit matrix can be only 0 or 1, we can use the bits of an integer to store up to 32 values per integer. In this example, if we imagine that the low-order bit is on the right, the first row can be stored as a single 32-bit integer 00000000-00000000-00000000-10110000, which has the decimal value of $128 + 32 + 16 = 176$. So if each row of the matrix is stored as a single integer where the bits of the integer are used to represent the presence or absence of an edge between nodes, the 9x9 matrix would require only 36 bytes.

In older programming languages, you'd have to implement a bit matrix from scratch using low-level bit operators such as left-shift, bitwise-or and so on. But the Microsoft .NET Framework System.Collections namespace has a BitArray type that makes implementing a program-defined BitMatrix type easy. A BitMatrix class can be defined as shown in Figure 4.

The BitMatrix class represents a square matrix and is essentially an array of BitArray array objects. I declare the BitMatrix class with private scope because I intend to embed it within a graph class definition rather than use it as a standalone class. The BitMatrix constructor accepts a parameter n that's the dimension of an $n \times n$ matrix, allocates a column of size n of BitArray array objects and then instantiates each BitArray using size n . Because there's no bit type in the .NET Framework, the values in a BitArray—and therefore in the BitMatrix class—are exposed as type bool, as you can see in the SetValue method. Notice that to keep my code short, I've removed normal error checking.

Using the BitMatrix could look like:

```
BitMatrix matrix = new BitMatrix(9);
matrix.SetValue(5, 8, true);
matrix.SetValue(8, 5, true);
bool connected = matrix.GetValue(2, 6);
```

The first line creates a 9x9 BitMatrix object initially set to all false (or zeros) to represent an unweighted, undirected graph with nine nodes. The second line sets row 5, column 8 to true/1 to indicate there's an edge between node 5 and node 8. The third line sets row 8, column 5 to true/1 so that the graph edge representation is consistent. The fourth line fetches the value at

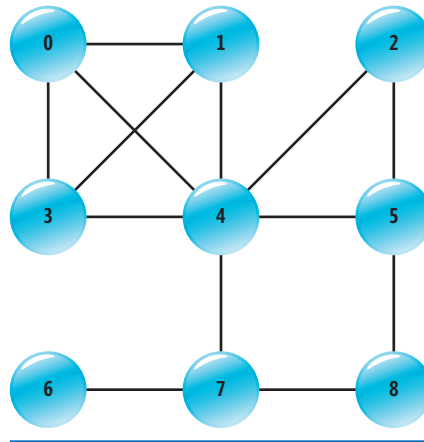


Figure 1 A Graph for the Maximum Clique Problem

row 2, column 6, a value indicating whether or not there's an edge between nodes 2 and 6, which would be false/0. Notice that determining whether or not two nodes are adjacent is just a quick array lookup.

A Graph Class

With a BitMatrix class in hand, it's easy to define an efficient graph class suitable for the maximum clique problem and many other graph-related problems. The structure of a graph class is presented in Figure 5. The graph class has dependencies on namespaces System, System.IO and System.Collections. For the example program, I placed the graph class directly inside the console app, but you may want to place the code in a class library.

The graph class definition starts with:

```
public class MyGraph
{
    private BitMatrix data;
    private int numberNodes;
    private int numberEdges;
    private int[] numberNeighbors;
    ...
}
```

I name the class MyGraph. It's somewhat tempting to try to define an all-purpose graph class, but there are so many variations of graphs that it's a better idea to define different graph classes for different kinds of problems. The graph class I define here is aimed at solving the maximum clique and related problems, so I could've named the class something like MaxCliqueGraph. The class has four data fields. The first is a BitMatrix object as described in the previous section. The numberNodes and numberEdges fields hold the number of nodes (nine in the example) and the number of undirected edges (13 in the example) in the graph.

```

C:\GraphMaxClique\bin\Debug>GraphMaxClique.exe

Begin graph for maximum clique demo

Validating DIMACS format graph file ..\..\DimacsGraph.clq
Graph file validation complete

Loading graph into memory

Validating graph data structure
Graph data structure validation complete

Graph adjacency representation:
0: 1 3 4
1: 0 3 4
2: 4 5
3: 0 1 4
4: 0 1 2 3 5 7
5: 2 4 8
6: 7
7: 4 6 8
8: 5 7

Are nodes 5 and 8 adjacent? True
Number neighbors of node 4 = 6

End graph for maximum clique demo
```

Figure 2 Graph Loading and Validation

When solving many graph problems, it's necessary to know how many neighbors a node has—that is, how many nodes are connected to a given node. For the example graph in **Figure 1**, node 5 has three neighbors. The number of neighbors a node has is also called the degree of the node. For a given node, this value could be computed on the fly when needed by counting the number of true/1 values in the node's data row. A much faster approach is to count and store the number of neighbors for each node once in the graph constructor and then do an array lookup when needed. So, for the example graph after instantiation, array `numberNeighbors` would have nine cells with values [3,3,2,3,6,3,1,3,2], indicating node 0 has three neighbors, node 1 has three neighbors, node 2 has two neighbors and so on.

The graph class constructor is:

```
public MyGraph(string graphFile, string fileFormat)
{
    if (fileFormat.ToUpper() == "DIMACS")
        LoadDimacsFormatGraph(graphFile);
    else
        throw new Exception("Format " + fileFormat + " not supported");
}
```

The constructor accepts a text file that holds graph data and a string that indicates the specific format of the data file. Here, I immediately transfer control to a helper method

	0	1	2	3	4	5	6	7	8
0	0	1	0	1	1	0	0	0	0
1	1	0	0	1	1	0	0	0	0
2	0	0	0	0	1	1	0	0	0
3	1	1	0	0	1	0	0	0	0
4	1	1	1	1	0	1	0	1	0
5	0	0	1	0	1	0	0	0	1
6	0	0	0	0	0	0	0	1	0
7	0	0	0	0	1	0	1	0	1
8	0	0	0	0	0	1	0	1	0

Figure 3 A Bit Matrix Graph Representation

`LoadDimacsFormatGraph`. This design allows the graph class to be easily extended to accommodate multiple data file formats. If you're a fan of enumeration types, the file format parameter can be implemented using an enumeration.

The heart of the `MyGraph` class is the `LoadDimacsFormatGraph` method, which reads a source data file and stores the graph representation. There are many more or less standard graph file formats. The one I use here is called the DIMACS format. The acronym DIMACS stands for Discrete Mathematics and Theoretical Computer Science. DIMACS is a collaborative association headed by Rutgers University.

The example program shown in **Figure 2** uses a file named `DimacsGraph.clq`, which is listed in **Figure 6**. Lines beginning with `c` are comment lines. There's a single line beginning with `p` that has the string "edge," followed by the number of nodes, followed by the number of edges. Lines beginning with `e` define edges. Notice that the DIMACS file format is blank-space delimited and 1-based, and that each edge is stored only once.

The load method begins:

```
private void LoadDimacsFormatGraph(string graphFile)
{
    FileStream ifs = new FileStream(graphFile, FileMode.Open);
    StreamReader sr = new StreamReader(ifs);
    string line = "";
    string[] tokens = null;
    ...
}
```

When reading text files, I prefer using the `FileStream` and `StreamReader` classes, but you may want to use one of the many .NET alternatives. Next:

```
line = sr.ReadLine();
line = line.Trim();
while (line != null && line.StartsWith("p") == false) {
    line = sr.ReadLine();
    line = line.Trim();
}
...
}
```

I perform a priming read and then advance to the `p` line in the data file. Because text files can easily acquire spurious whitespace characters over time, I use the `Trim` method to help avoid problems. Continuing:

```
tokens = line.Split(' ');
int numNodes = int.Parse(tokens[2]);
int numEdges = int.Parse(tokens[3]);
sr.Close(); ifs.Close();
this.data = new BitMatrix(numNodes);
...
}
```

I use the `String.Split` method to parse the `p` line. At this point, `tokens[0]` holds the string literal "p," `tokens[1]` holds "edge," `tokens[2]` holds "9" and `tokens[3]` holds "13." I use the `int.Parse` method (I could've used `Convert.ToInt32`) to convert the number of nodes and edges into `int` values that I store in local variables `numNodes` and `numEdges`. I could've stored these values into class fields `this.numberNodes` and `this.numberEdges` at this time. Now that I've determined the number of nodes and the number of edges, I close the data file and instantiate the `BitMatrix` data field.

Figure 4 A BitMatrix Class

```
private class BitMatrix
{
    private BitArray[] data;
    public readonly int Dim;

    public BitMatrix(int n)
    {
        this.data = new BitArray[n];
        for (int i = 0; i < data.Length; ++i) {
            this.data[i] = new BitArray(n);
        }
        this.Dim = n;
    }

    public bool GetValue(int row, int col)
    {
        return data[row][col];
    }

    public void SetValue(int row, int col, bool value)
    {
        data[row][col] = value;
    }

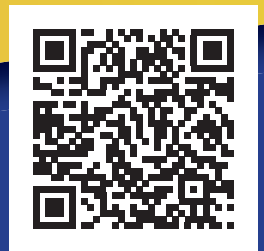
    public override string ToString()
    {
        string s = "";
        for (int i = 0; i < data.Length; ++i) {
            for (int j = 0; j < data[i].Length; ++j) {
                if (data[i][j] == true)
                    s += "1 ";
                else
                    s += "0 ";
            }
            s += Environment.NewLine;
        }
        return s;
    }
}
```

WINDOWS FORMS

WORD PROCESSING COMPONENTS REALLY? FREE?

TX Text Control Express
is your FREE - free as in beer -
RichTextBox replacement
for Visual Studio®

Download at: www.textcontrol.com/express



Visual Studio is a trademark of Microsoft Corporation in the United States and/or other countries.



Word Processing Components
for Windows Forms, WPF & ASP.NET

WWW.TEXTCONTROL.COM



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

Figure 5 A Graph Class Definition

```
public class MyGraph
{
    private BitMatrix data;
    private int numberNodes;
    private int numberEdges;
    private int[] numberNeighbors;

    public MyGraph(string graphFile, string fileFormat)
    {
        if (fileFormat.ToUpper() == "DIMACS")
            LoadDimacsFormatGraph(graphFile);
        else
            throw new Exception("Format " + fileFormat + " not supported");
    }

    private void LoadDimacsFormatGraph(string graphFile)
    {
        // Code here
    }

    public int NumberNodes
    {
        get { return this.numberNodes; }
    }

    public int NumberEdges
    {
        get { return this.numberEdges; }
    }

    public int NumberNeighbors(int node)
    {
        return this.numberNeighbors[node];
    }

    public bool AreAdjacent(int nodeA, int nodeB)
    {
        if (this.data.GetValue(nodeA, nodeB) == true)
            return true;
        else
            return false;
    }

    public override string ToString()
    {
        // Code here
    }

    public static void ValidateGraphFile(string graphFile, string fileFormat)
    {
        if (fileFormat.ToUpper() == "DIMACS")
            ValidateDimacsGraphFile(graphFile);
        else
            throw new Exception("Format " + fileFormat + " not supported");
    }

    public static void ValidateDimacsGraphFile(string graphFile)
    {
        // Code here
    }

    public void ValidateGraph()
    {
        // Code here
    }

    // -----
    private class BitMatrix
    {
        // Code here
    }
    // -----
} // Class MyGraph
```

Now I'm ready to read edge data from the data file:

```
ifs = new FileStream(graphFile, FileMode.Open);
sr = new StreamReader(ifs);
while ((line = sr.ReadLine()) != null) {
    line = line.Trim();
    if (line.StartsWith("e") == true) {
        tokens = line.Split(' ');
        int nodeA = int.Parse(tokens[1]) - 1;
        int nodeB = int.Parse(tokens[2]) - 1;
        data.SetValue(nodeA, nodeB, true);
        data.SetValue(nodeB, nodeA, true);
    }
}
sr.Close(); ifs.Close();
...
```

I reopen the file and start reading from the beginning. Technically—because of the presence of the p line before any e lines—there's no need to use two reads of a DIMACS format file. However, for other file formats that don't explicitly store the number of edges, you may want to perform a double scan like the one used here. When the code encounters an e line such as "e 3 6", I parse the e line, convert the two nodes to type int and subtract 1 to change the representation from 1-based to 0-based. I use the SetValue method to create symmetric entries in the BitMatrix. Note that because the BitMatrix is symmetric, I could've stored just the upper or lower triangular portion to reduce memory.

Next, I take care of the numberNeighbors array:

```
this.numberNeighbors = new int[numNodes];
for (int row = 0; row < numNodes; ++row) {
    int count = 0;
    for (int col = 0; col < numNodes; ++col) {
        if (data.GetValue(row, col) == true) ++count;
    }
    numberNeighbors[row] = count;
}
...
```

For each node, I walk across its corresponding row and count the number of true/1 values that gives the number of edges and therefore the number of neighbors the node has. The LoadDimacsFormatGraph method finishes with:

```
...
this.numberNodes = numNodes;
this.numberEdges = numEdges;
return;
}
```

After transferring the number of nodes and the number of edges from local variables to class field variables, I use an explicit return for readability to exit the method.

The rest of the MyGraph class is easy. I expose the private numberNodes and numberEdges class fields as read-only values using the C# class Property mechanism:

```
public int NumberNodes {
    get { return this.numberNodes; }
}
```

```
public int NumberEdges {
    get { return this.numberEdges; }
}
```

I prefer using the explicit Property syntax, but you can use auto-implemented property syntax if you're using .NET 3.0 or greater. I expose the number of neighbors a node has through a method:

```
public int NumberNeighbors(int node) {
    return this.numberNeighbors[node];
}
```

When working with graphs, it's difficult to know when to perform standard error checking and when to omit checking. Here, I don't check to see if the node parameter is in the range 0 .. this.numberNodes-1, leaving me open to an array index out of

PRECISELY PROGRAMMED FOR SPEED

DynamicPDF—Comprehensive PDF Solutions for .NET Developers

ceTe Software's DynamicPDF products provide real-time PDF generation, manipulation, conversion, printing, viewing, and much more. Providing the best of both worlds, the object models are extremely flexible but still supply the rich features you need as a developer. Reliable and efficient, the high-performance software is easy to learn and use. If you do encounter a question with any of our components, simply contact ceTe Software's readily available, industry-leading support team.



DynamicPDF

[WWW.DYNAMICPDF.COM](http://www.DynamicPDF.com)



TRY OUR PDF SOLUTIONS FREE TODAY!

www.DynamicPDF.com/eval or call 800.631.5006 | +1 410.772.8620

ceTe software

Figure 6 DIMACS Format Data File

```
c DimacsGraph.clq
c number nodes, edges: 9, 13
p edge 9 13
e 1 2
e 1 4
e 1 5
e 2 4
e 2 5
e 3 5
e 3 6
e 4 5
e 5 6
e 5 8
e 6 9
e 7 8
e 8 9
```

range exception. I usually add error-checks during development, then after development I remove those checks that I feel can safely be omitted in order to improve performance. Because of my data structure design with a `BitMatrix` class, writing a method to determine if two nodes are adjacent is easy:

```
public bool AreAdjacent(int nodeA, int nodeB)
{
    if (this.data.GetValue(nodeA, nodeB) == true)
        return true;
    else
        return false;
}
```

Recall that the `BitMatrix` is symmetric, so I can check either `GetValue(nodeA, nodeB)` or `GetValue(nodeB, nodeA)`. As I mentioned earlier, checking node adjacency dominates the runtime of many graph algorithms. When using a bit matrix, checking node adjacency is quick because the check is just an array lookup plus a little bit-manipulation overhead handled by the `BitArray` class.

I code a simple `ToString` method for the `MyGraph` class:

```
public override string ToString()
{
    string s = "";
    for (int i = 0; i < this.data.Dim; ++i) {
        s += i + ": ";
        for (int j = 0; j < this.data.Dim; ++j) {
            if (this.data.GetValue(i, j) == true)
                s += j + " ";
        }
        s += Environment.NewLine;
    }
    return s;
}
```

In the maximum clique scenario, performance isn't a big issue, so for the `ToString` method, I use simple string concatenation rather than the more efficient `StringBuilder` class. Here, I use `i` to index into the `BitMatrix` rows and `j` to index into the columns. I terminate the string with an `Environment.NewLine` rather than `"\n"` to make the `MyGraph` class more portable.

Validating the Graph

If you refer back to **Figure 2**, you'll notice that I perform two important types of graph validation: validating the graph data file before the graph object is instantiated and validating the internal graph representation after instantiation.

A complete discussion of graph validation and testing would require an entire article, so I'll just present an overview. You can obtain and view the complete validation code from code.msdn.microsoft.com/mag201110TestRun.

I perform data file validation using a static method `ValidateGraphFile` as shown in **Figure 5**. As with the `MyGraph` constructor, `ValidateGraphFile` immediately calls a helper method `ValidateDimacsGraphFile` to do the actual work. The file validation code iterates through the file to see if every line is in valid DIMACS form:

```
if (line.StartsWith("c") == false &&
    line.StartsWith("p") == false &&
    line.StartsWith("e") == false)
    throw new Exception("Unknown line type: " + line);
```

The method also checks the format of non-comment lines by attempting to parse. For example, for the single `p` line:

```
try {
    if (line.StartsWith("p")) {
        tokens = line.Split(' ');
        int numNodes = int.Parse(tokens[2]);
        int numEdges = int.Parse(tokens[3]);
    }
    catch {
        throw new Exception("Error parsing line = " + line);
    }
}
```

The method uses similar logic to test `e` lines. This pattern holds in general when validating graph data files: check for valid lines and attempt to parse data lines.

Once instantiated, I validate the internal graph representation using a method `ValidateGraph`. It turns out that a complete check of the graph data structure is surprisingly complex, so in practice it's common to check for just those errors that are most likely to occur. A common error in graph data files is a missing data line that creates an asymmetric `BitMatrix` data store. It can be checked with code like this:

```
for (int i = 0; i < this.data.Dim; ++i) {
    for (int j = 0; j < this.data.Dim; ++j) {
        if (this.data.GetValue(i, j) != this.data.GetValue(j, i))
            throw new Exception("Not symmetric at " + i + " and " + j);
    }
}
```

Other errors to check for include the presence of a `true/1` on the bit matrix main diagonal, a bit matrix consisting of either all `false/0` or `true/1`, and the sum of the values in the `numberNeighbors` array field not equaling the total number of `true/1` values in the bit matrix.

Stay Tuned for More Details

This article presented a graph data structure type that can be used for solving many graph-related problems including the maximum clique problem. The essential feature of the graph data structure is the use of a program-defined bit matrix that's efficient in terms of memory usage and that allows quick node-adjacency lookups. The bit matrix field of the graph data structure is implemented using the .NET `BitArray` class, which takes care of all the low-level bit manipulation operations. In the next Test Run column, I'll describe the maximum clique problem in more detail and show you a greedy algorithm solution that uses the graph structure described here. ■

DR. JAMES MCCAFFREY works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He's worked on several Microsoft products, including *Internet Explorer* and *MSN Search*. Dr. McCaffrey is the author of *".NET Test Automation Recipes"* (Apress, 2006), and can be reached at jammc@microsoft.com.

THANKS to the following technical experts for reviewing this article:
Paul Koch, Dan Liebling, Ann Loomis Thompson and Shane Williams

Does your Team do more than just track bugs?

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

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



Alexsys Team

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

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

Native Smart Card Login Support including Government and DOD



New in Team 2.11

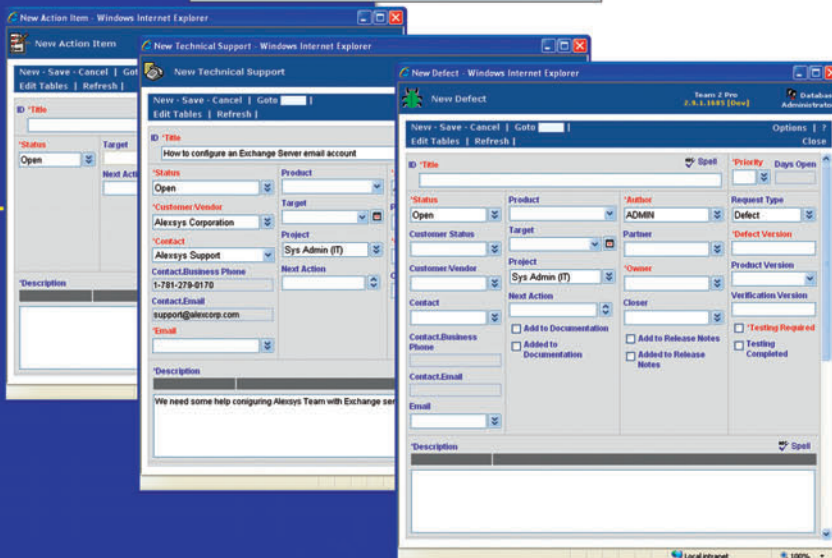
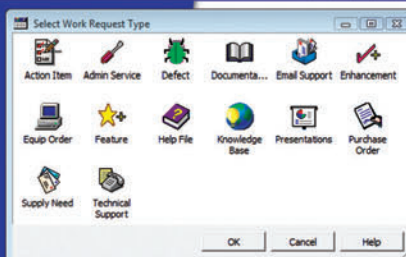
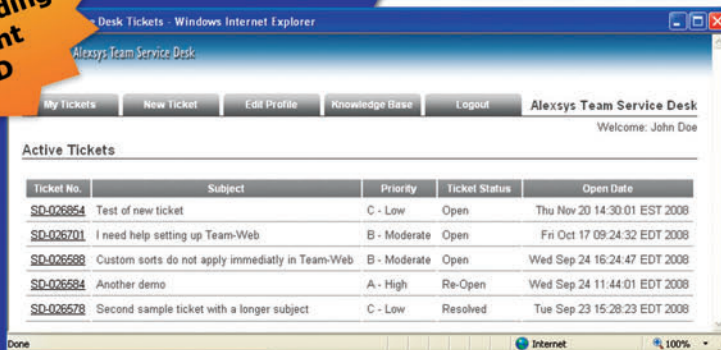
- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

Service Desk Features

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

Team 2 Features

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



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.



Pages and Pop-ups in Windows Phone 7

The interaction between user and computer application normally proceeds without interruption. We type text into the word processor; enter numbers and formulae into the spreadsheet; and flip the pages in an e-book reader. But occasionally, the application requires additional information from the user, or the user initiates an operation specifically to provide the application with other information.

In a traditional Windows application, we all know what happens next: A dialog box appears. Fill it out and click OK, or Cancel if you've changed your mind.

What happens in a Windows Phone 7 application isn't quite clear, however. Neither in Silverlight for the Web nor Silverlight for Windows Phone does there exist anything called a "dialog box." I still like the term to describe any mechanism that allows the application to interrogate the user for information, but obviously Silverlight dialog boxes are a little different from the traditional kind.

Over the past several installments of this column, I've been progressively building an e-book reader for Windows Phone 7 based on plain-text book files downloaded from Project Gutenberg. It has come time to enhance that program with a whole bunch of dialog boxes.

Two Approaches

In implementing dialog boxes in a Silverlight application for Windows Phone 7, you have two options.

Perhaps the most obvious approach is to define the dialog box as a separate page. Derive a new class from `PhoneApplicationPage` and populate it with a bunch of buttons and text boxes and whatnot. A page then invokes this dialog box by navigating to it using the page's `NavigationService` object. The Back button (or something else) terminates the dialog box and goes back to the page that invoked it.

The second option is what I call the "pop-up" approach. This approach could make use of the `Popup` class, but it doesn't need to. (I'll discuss the difference shortly.) This kind of dialog box generally derives from `UserControl` and appears visually on top of the page that invokes it, and then disappears when the user is finished interacting with it. No navigation is involved.

Dialog boxes implemented as navigable pages are inherently modal. (A modal dialog box inhibits interaction with the window or page that invoked it until the dialog box is dismissed.) With the pop-up approach, a dialog box can be either modal or modeless. Implementing a modal pop-up requires the programmer to make sure that

the user can't interact with the underlying page while the pop-up is active. In this sense, it's a little easier to implement a modeless pop-up, but typically the process of juggling simultaneous input from a modeless dialog and an underlying page can be tricky.

Beginning with the `MiddlemarchReader` program that I presented in my July column (msdn.microsoft.com/magazine/hh288085), my e-book readers have included an option to display a list of chapters in a `ListBox`. You can then select one of the chapter titles to jump to the beginning of that chapter. This qualifies as a dialog box, and I chose to implement it as a pop-up even though the logic was somewhat messy: The `AppBar` button that invoked the pop-up also served to dismiss the pop-up, and that required a somewhat different image for the button.

Neither in Silverlight for the Web nor Silverlight for Windows Phone does there exist anything called a "dialog box."

In retrospect, I should have implemented this dialog box as a separate page, and that's how it's done in the `HorrorReader` program I'll be describing in this article. To celebrate October and Halloween, `HorrorReader` lets you read four horror classics: "Frankenstein," "Dracula," "Dr. Jekyll and Mr. Hyde" and "The Turn of the Screw," as shown in **Figure 1**. (The e-book reader next month will finally expand the potential library to some 35,000 books, I promise!)

In changing the chapters `ListBox` from a pop-up to a navigable page, I searched deep into my heart to try to understand why I've tended to implement Windows Phone 7 dialog boxes as pop-ups. I've discovered the surprising motivation: Fear.

Fear of Navigation

If you've done any Windows Phone 7 programming at all, you probably know about tombstoning: In certain circumstances, a running application can be terminated and removed from memory. This happens when the user presses the Start button on the phone to view the start screen; the phone hasn't received any input for a while and turns off its display to go into a locked condition, or the user turns off the screen manually.

Code download available at code.msdn.microsoft.com/mag201110UIFrontiers.

The application is restarted when the user unlocks the screen or presses the Back button to navigate back to the program. The Windows Phone 7 OS will display the last active page in the application, but the restoration of any other information is entirely the programmer's responsibility. Generally, a page uses the State dictionary of PhoneApplicationPage to save transient information associated with the page, while the App object uses the State dictionary of PhoneApplicationService to store transient application data, and isolated storage for permanent data.

In coding for tombstoning, programmers override the OnNavigatedFrom method in the page to save page information and OnNavigatedTo to restore this information. That's fine when the program is being tombstoned or being revived after being tombstoned. But if the page is simply in the process of navigating to another page, or returning from that navigation, then saving and restoring page information is unnecessary. Depending on the amount of information involved, this extra activity could slow up page navigation significantly.

One way to avoid this extra activity is simply to restrict the application to one page and implement dialog boxes with pop-ups! That's what I did in the previous versions of the e-book reader. I avoided page navigation because I feared slowing it down with tombstoning code.

But that's silly. It's really only a problem if tombstoning is implemented in what I now think of as "the dumb approach"! A page shouldn't save a lot of state information unless the application is really being tombstoned, and it shouldn't attempt to restore that information unless it's being revived from a tombstoned state. Windows Phone 7.1 will provide the navigation overrides with additional information to more intelligently implement these methods. Meanwhile, you can relegate all heavy lifting to the App class, which is primarily responsible for handling the events implemented by PhoneApplicationService. These events truly indicate if an application is being tombstoned or revived.

It's usually beneficial to tighten up the OnNavigatedTo override as well. This logic can be rather simple: If a particular field is null, it needs to be regenerated; if it isn't null, then the object is the same as it was before navigation because the application wasn't tombstoned.

The Structure of HorrorReader

The App file in HorrorReader has two public properties that reference objects stored in isolated storage. The first, AppSettings, stores application settings that apply to all the books. These include the font family, font size and the style of page transition. The second, the CurrentBook property of App, is an object of type BookInfo, which has all the individual book-related properties, including the filename of the actual book, the current chapter and page, the collection of ChapterInfo objects storing pagination data, and collections of bookmarks and annotations, which are new with this version. Each of the four books in the library has its own BookInfo object stored in isolated storage, but this BookInfo object isn't created until you first read the book.

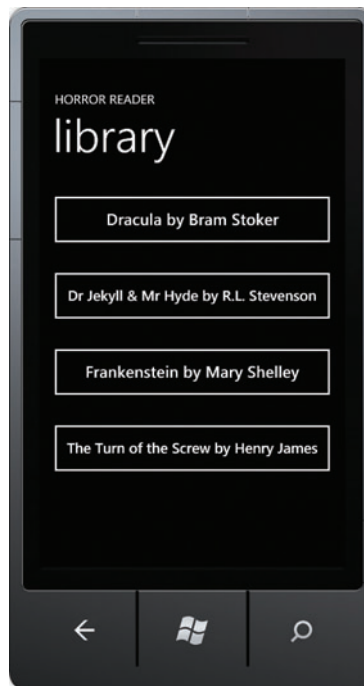


Figure 1 The MainPage Display of HorrorReader



Figure 2 The Settings Pivot for Font Selection

HorrorReader has six classes that derive from PhoneApplicationPage. These classes are all part of the HorrorReader project and are easily identified with the word "Page" in their names. Aside from App, all other classes are in the Petzold.Phone.EBookReader dynamic link library.

As you've seen in **Figure 1**, MainPage has four buttons that let you select one of the four available books. (This page will be replaced in next month's version of the program.) When the user clicks one of these buttons, MainPage navigates to BookViewerPage, which is primarily responsible for hosting the BookViewer control and implementing several pop-ups.

Dialog boxes implemented as navigable pages are inherently modal.

BookViewerPage has four ApplicationBar buttons. The first three cause navigation to other pages: ChaptersPage, BookmarksPage and AnnotationsPage. A Bookmark is simply a reference to a particular page in the book with a label entered by the user. An Annotation is a text selection accompanied by an optional note. Each of these three pages contains a ListBox, and each of them causes the BookViewerPage to jump to a new page in the book.

The ApplicationBar menu in BookViewerPage has three items: "smaller text," "larger text" and "settings." The first two cause changes to the font size in increments of 10 percent, and "settings" navigates to SettingsPage, which implements a Pivot control for selecting fonts and the desired page transition, as shown in **Figure 2**. The text at the bottom of the page is a preview of the selected font.

In working on this page, I ran into an apparent conflict with the Slider and the GestureListener class from the Windows Phone Toolkit. I had to implement my own Slider, and in the process made it jump only in 10 percent increments.

I'm not entirely happy with the use of PhoneApplicationPage derivatives as dialog boxes. I prefer a more structured way to pass information to a dialog box and get information back. This transfer of data often becomes somewhat clumsy within a navigational structure. For example, I would've preferred for the BookViewerPage to pass a reference of the current book to ChaptersPage, and for ChaptersPage to return the chapter selected by the user back to BookViewerPage. In other words, I want ChaptersPage to work more like a function without side effects. Perhaps someday I'll design a wrapper around PhoneApplicationPage that lets me do this. Meanwhile, for HorrorReader, I simply have all the pages share data by referencing the same CurrentBook property in the App class.

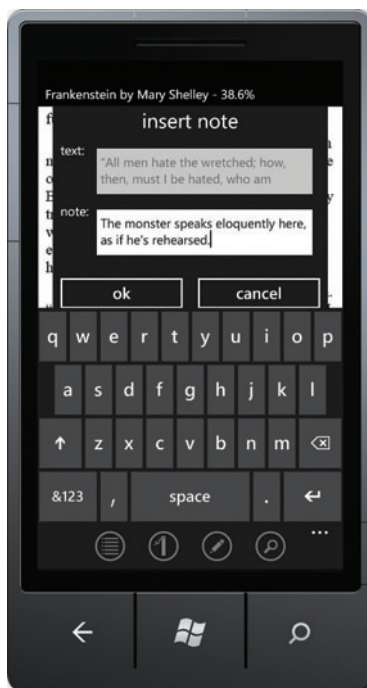


Figure 3 The AnnotationDialog Pop-Up

Pop up or Pop-Up?

Although I went with PhoneApplicationPage derivatives for the chapters list, bookmarks list, annotations list and settings, I still needed several pop-ups.

There are basically two ways to implement a pop-up in Silverlight for Windows Phone. One approach is simply to put a control (or, more commonly, a UserControl derivative) right in

There are basically two ways
to implement a pop-up in
Silverlight for Windows Phone.

the page's visual tree. This control sits on top of everything else, but its Visibility property is initialized to Collapsed. When the pop-up needs to pop up, simply set the Visibility property to Visible.

For a modal pop-up, you'll also want to disable everything else on the page. You can do this by setting the IsEnabled property of the underlying control to false, or alternatively setting the IsHitTestVisible property to false. These two properties are similar, but IsEnabled is restricted to controls while IsHitTestEnabled can also be used with FrameworkElement derivatives such as panels. The IsEnabled property also causes some controls to dim out. Another option is to make the pop-up full screen with a translucent background that blocks touch input. Regardless what technique you use, you'll probably also need to disable the ApplicationBar. (More on this little problem shortly.)

An alternative is to use the Popup element. The Popup element has a Child property that you'll probably set to a UserControl

derivative. (I wish I could derive from Popup, but it's sealed.) By default, the IsOpen property of Popup is false, and you set it to true to make the child of the Popup visible. Popup also has convenient HorizontalOffset and VerticalOffset properties that let you position the child.

One interesting aspect of Popup is that it doesn't require a parent element. In other words, it doesn't have to be part of the visual tree. You can simply create a Popup object in code, set the Child property and set the IsOpen property to true and it will appear visually on top of everything else.

Watch out, though. If Popup has no parent, the HorizontalOffset and VerticalOffset properties are relative to the upper-left corner of the PhoneApplicationFrame object, which conceptually underlies all the PhoneApplicationPage objects. The frame includes the system tray at the top of the screen, but the pop-up won't appear on top of the system tray. If the system tray is visible, the Popup child will be clipped at the top. Only the part overlaying the PhoneApplicationPage will be visible. You'll want to set the VerticalOffset property to a

non-zero value to accommodate the system tray.

The advantage of a Popup with no parent is that it's easy to make it modal. You simply set the IsEnabled property of the PhoneApplicationPage derivative to false and effectively disable everything on the page in one shot.

Gosh, wouldn't that be nice? In reality, it doesn't happen, because the ApplicationBar isn't part of the visual tree of the page. You'll need to disable the ApplicationBar separately. It has a handy IsMenuEnabled property, but it doesn't have a single property to disable the buttons. You'll have to handle those buttons individually.

For two of the pop-ups I needed, I wanted to move them to various places relative to the page. That's easy when the dialog is a child of Popup. Otherwise, you'll need a TranslateTransform on the dialog control. To simplify tombstoning, I decided to be consistent and use the Popup element for all my pop-ups. I also decided to define each Popup and child in the visual tree of the page. The HorizontalOffset and VerticalOffset properties are thus relative to the page rather than to the frame.

One of these pop-ups is a ListBox named "textSelectionMenu," defined in BookViewerPage.xaml. This is the menu that appears when you make a text selection, as I described in my September column (msdn.microsoft.com/magazine/hh394142). One of the options is "note," which invokes another pop-up defined by a UserControl derivative named AnnotationDialog. This lets the user type in a note, as shown in Figure 3.

You can define a new bookmark by flicking your finger up or down the page. The BookmarkDialog lets you enter a label.

BookmarkDialog and AnnotationDialog can also appear, respectively, on BookmarksPage and AnnotationsPage for editing an existing item. You can also delete a bookmark or note on these

pages, and that invokes another pop-up called `OkCancelDialog`. This is similar to the standard `MessageBox` available in Windows Phone 7, except it doesn't make a noise when it appears.

The final pop-up is `FindDialog`, invoked by the fourth `AppBar` button and shown in **Figure 4**. This lets you search for text in the book with familiar options. As each match is found, the text is highlighted and the `FindDialog` jumps to the top or bottom of the page to let the highlighted text be visible.

Tombstoning Pop-Ups

Should pop-ups accommodate tombstoning? Here's the scenario I posed to myself: Suppose I'm searching for some text in a book using the pop-up dialog shown in **Figure 4**. Then I set the phone down on the table. Several minutes later, I pick up the phone and the screen is locked. I push the On button and sweep up the wallpaper. Should the `FindDialog` still be visible in the state I left it?

My reluctant answer was: of course. I had to acknowledge that this is what the user expects and this is what should happen. That meant that pop-ups needed to be tombstoned.

So, in any page that hosts a `Popup`, I defined a field of type `Popup` named `activePopup`, which is set whenever a pop-up is visible. If this field is set during a call to `OnNavigationFrom`, it can only mean that the program is being tombstoned, because everything else is disabled while the pop-up is visible. In that case, the page saves the name of the `Popup` in the page's `State` dictionary. It also checks if the child of the `Popup` implements the `ITombstonable` interface, which I defined with two methods: `SaveState` and `RestoreState`. This is how the `AnnotationDialog`, `BookmarkDialog`, `OkCancelDialog` and `FindDialog` dialog pop-ups save and restore their current states during tombstoning.

As I was implementing various dialog boxes in the form of pop-ups, it became obvious to me that I needed to override the behavior of the Back button rather frequently.

Back Button Overrides

I wrote an entire book on Windows Phone 7 programming and I didn't include even one example of overriding the `OnBackKeyPress` method defined by `PhoneApplicationPage`. Silly me.

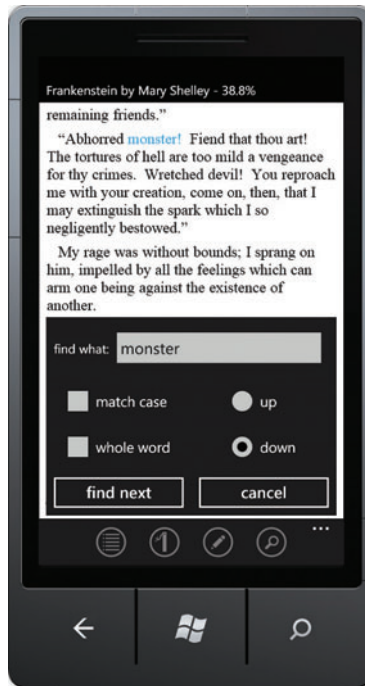


Figure 4 The `FindDialog` Pop-Up

The Back key referred to in this method is the leftmost of the three hardware buttons on the phone. By default, the Back button navigates from a page back to the page that invoked it. If an application is on its main page, then the Back button terminates the program.

As I was implementing various dialog boxes in the form of pop-ups, it became obvious to me that I needed to override the behavior of the Back button rather frequently. The general rule is this: Whenever it's conceivable that the user will push the Back button without desiring to navigate away from the page or terminate the application, you should override `OnBackKeyPress`.

Any page that can host pop-ups—and in my case that was `BookViewerPage`, `AnnotationsPage` and `BookmarksPage`—should override the Back key to dismiss the pop-up, much like clicking the window Close button in a traditional dialog box. The argument to the `OnBackKeyPress` method is an instance of `CancelEventArgs`; set the `Cancel` property to true to inhibit the Back key from performing its normal navigational function.

If a pop-up is active and a `TextBox` on the pop-up has the input focus, then an on-screen keyboard will be displayed. Pressing the Back key at this time automatically dismisses the keyboard. Pressing Back again should dismiss the pop-up. Press Back again and you'll navigate back to the previous page. In all cases, a dramatic visual change gives good feedback to the user that pressing the Back key actually did something.

Watch out when overriding `OnBackKeyPress`. It should never be possible for the user to get stuck in a loop where successive presses of the Back button don't terminate the application! You'll never get the program into the Windows Phone 7 marketplace if this is the case.

Ready for the Front End

In programming, there's often a big difference between supporting one instance of some entity and supporting more than one instance. Besides the various pages and pop-ups in `HorrorReader`, the program has made a big leap over my previous e-book readers in letting you read four books rather than just one. Each book has its own `BookInfo` object that's stored in isolated memory, so each book is independent of all the rest.

Now all that's necessary is to replace `MainPage` with a new front end that lets you download books from the Project Gutenberg site. No other changes should be necessary. That should be simple, right? ■

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. This month marks the 25th anniversary of Petzold's contributions to MSDN Magazine and its predecessor, Microsoft Systems Journal.

THANKS to the following technical expert for reviewing this article:
Richard Bailey

5 CODE-FILLED DAYS IN FLORIDA!



INTENSE TRAINING + AN AWESOME LOCATION:

Visual Studio Live! Orlando is a must-attend event for **developers**, **software architects** and **designers** that provides the tools, technologies and tips you need to solve development challenges. Educational tracks lead by IT experts and .NET rock stars include:

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

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

Use Promo Code **OCTAD**

**PICK YOUR
SESSIONS**

PLATINUM SPONSORS

Microsoft

Syncfusion
Deliver innovation with ease

SUPPORTED BY

Microsoft
Visual Studio

msdn

Visual Studio
MAGAZINE

PRODUCED BY

1105 MEDIA

VSLIVE.COM/ORLANDO

Use Promo Code **OCTAD**

VISUAL STUDIO LIVE! ORLANDO

Silverlight / WPF	Developing Services	Windows Phone 7	Cloud Computing	Data Management	Programming Practices	Web / HTML5	Visual Studio 2010 / .NET 4
-------------------	---------------------	-----------------	-----------------	-----------------	-----------------------	-------------	-----------------------------

Visual Studio Live! Pre-Conference Workshops: Monday, December 5, 2011 (Separate entry fee required)

MWK1 Workshop: SQL Server for Developers <i>Andrew Brust & Leonard Lobel</i>	MWK2 Workshop: Making Effective Use of Silverlight and WPF <i>Billy Hollis & Rockford Lhotka</i>	MWK3 Workshop: Programming with WCF in One Day <i>Miguel Castro</i>
--	--	---

Visual Studio Live! Day 1: Tuesday, December 6, 2011

Keynote *Microsoft TBA*

T1 HTML5 & CSS3 Mini-Bootcamp for ASP.NET Developers <i>Todd Anglin</i>	T2 Intense Intro to Silverlight <i>Billy Hollis</i>	T3 AppFabric, Workflow and WCF - The Next Generation Middleware <i>Ron Jacobs</i>	T4 So Many Choices, So Little Time: Understanding Your .NET 4 Data Access Options <i>Lenni Lobel</i>
T5 The HTML5 Mullet: Form Input and Validation <i>Todd Anglin</i>	T6 XAML: Achieving Your Moment Of Clarity <i>Miguel Castro</i>	T7 What's New in WCF 4 <i>Ido Flatow</i>	T8 Microsoft Session <i>TBA</i>

Birds-of-a-Feather Lunch & Visit Exhibits

T9 Chalk Talk: What's New and Cool in Silverlight 5 <i>Pete Brown</i>	T10 Chalk Talk: Building Applications Using CSLA .NET <i>Rockford Lhotka</i>	T11 Learning MVC - for the Web Forms Developer <i>Adam Tuliper</i>	T12 Fundamental Design Principles for UI Developers <i>Billy Hollis</i>
T13 Creating Scalable Stateful Services Using WCF and WF <i>Marcel de Vries</i>	T14 Microsoft Session <i>TBA</i>	T15 MVC, Razor, and jQuery - The New Face of ASP.NET <i>Ido Flatow</i>	T16 Bind Anything to Anything in XAML <i>Rockford Lhotka</i>
T17 AppFabric Caching: How It Works and When You Should Use It <i>Jon Flanders</i>	T18 Microsoft Session <i>TBA</i>		

Visual Studio Live! Day 2: Wednesday, December 7, 2011

Keynote *Scott Cate*

W1 Creating a Data Driven Web Site Using WebMatrix and ASP.NET Razor <i>Rachel Appel</i>	W2 Javascript and CSS 3 Patterns for HTML5 <i>John Papa</i>	W3 If Not IaaS, When Should I Use Windows Azure VM Role? <i>Eric D. Boyd</i>	W4 Team Foundation Server Build Automation Inside Out <i>Marcel de Vries</i>
W5 Hack Proofing Your ASP.NET Web Forms and MVC Applications <i>Adam Tuliper</i>	W6 A Lap Around WPF v.next <i>Pete Brown</i>	W7 What is Windows Azure Marketplace DataMarket? <i>Michael Stiefel</i>	W8 Implementing Custom Shells, Silverlight Custom Controls and WCF RIA <i>Michael Washington</i>

Lunch & Visit Exhibits

W9 Chalk Talk: Advanced Patterns with MVVM in Silverlight and Windows Phone 7 <i>John Papa</i>	W10 Chalk Talk: Building RESTful Services with WCF <i>Jon Flanders</i>	W11 How Orchard CMS Works <i>Rachel Appel</i>	W12 Light Up on Windows 7 with Silverlight and WPF <i>Pete Brown</i>
W13 Deciding Between Relational Databases and Tables in the Cloud <i>Michael Stiefel</i>	W14 BI in the Cloud with SQL Azure Reporting <i>Eric D. Boyd</i>	W15 HTML5 and Internet Explorer 9: Developer Overview <i>Ben Hoelting</i>	W16 Radically Advanced Templates for WPF and Silverlight <i>Billy Hollis</i>
W17 Windows Azure Platform Overview <i>Vishwas Lele</i>	W18 Overview of Project 'Crescent' <i>Andrew Brust</i>		

Wild Wednesday with Developer Duel

Visual Studio Live! Day 3: Thursday, December 8, 2011

TH1 Getting Started with ASP.NET MVC <i>Philip Japikse</i>	TH2 XNA Games for Windows Phone 7 <i>Brian Peek</i>	TH3 Building Windows Azure Applications <i>Vishwas Lele</i>	TH4 REST with Silverlight 5, WCF Web API, and a Little ASP.NET MVC3 <i>Pete Brown</i>
TH5 Test Driving ASP.NET MVC <i>Philip Japikse</i>	TH6 Working with Data on Windows Phone 7 <i>Sergey Barskiy</i>	TH7 Building Compute-Intensive Apps in Windows Azure <i>Vishwas Lele</i>	TH8 Session <i>TBA</i>
TH9 Busy Developer's Guide to (ECMA/Java)Script <i>Ted Neward</i>	TH10 Building Native Mobile Apps with HTML5 & jQuery <i>Jon Flanders</i>	TH11 Building and Running the Windows Azure Developer Portal <i>Chris Mullins</i>	TH12 Multi-Touch Madness! <i>Brian Peek</i>

Lunch

TH13 Using Code First (Code Only) Approach with the Entity Framework <i>Sergey Barskiy</i>	TH14 Getting Started with Windows Phone 7 <i>Scott Golightly</i>	TH15 The LINQ Programming Model <i>Marcel de Vries</i>	TH16 Application Lifecycle Management and Visual Studio: What's Next <i>Brian Randell</i>
TH17 Using MEF to Develop Composable Applications <i>Ben Hoelting</i>	TH18 Data Binding and MVVM Patterns in HTML5 <i>John Papa</i>	TH19 Microsoft Session <i>TBA</i>	TH20 Visual Studio v.Next <i>Brian Randell</i>

Visual Studio Live! Post-Conference Workshops: Friday, December 9, 2011 (Separate entry fee required)

FWK1 Workshop: Architectural Katas <i>Ted Neward</i>	FWK2 Workshop: ALM in 2011: Visual Studio 2010 and the Next Big Release <i>Brian Randell</i>
--	--

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

Sessions and speakers are subject to change.



Imagine That

I just returned from judging the world finals of the Microsoft Imagine Cup, a programming contest for college students. The experience was transcendent.

I'd been a judge at the first worldwide finals, in Barcelona in 2003, and had written in "Why Software Sucks" about my experiences there. The contest has grown enormously since then, from 1,000 student entrants in regional- and country-level tourneys to more than 350,000 today. The finals have expanded from 16 teams in one division to 128 teams (yes, we geeks like our powers of two) across nine divisions; and evolved from an afterthought at Tech-Ed to its own separate conference, taking over the Times Square Marriott hotel in NYC.

The teams came from all over: stalwarts such as Japan and France, the BRIC countries (Brazil, Russia, India, China) nipping at their heels, upcomers like Vietnam and Bangladesh nipping at theirs. Even sub-Saharan Africa is starting to enter, with teams from Uganda and Senegal.

The best projects are scary smart: Team Hermes from Ireland, with a system to help teenage drivers kill fewer of themselves and others; Team Note-Taker from the United States, with a note-taker system for visually handicapped students; Team Oasys from Jordan, with an assistance device that lets quadriplegics control PCs by moving their heads.

The raw brain power here is stunning. The ingenuity, the imagination, the incredible things that are possible when you haven't yet learned what's impossible. And also the blindness, the naiveté, the solving of the wrong problems when you haven't yet learned what's important and what isn't, or what users really do rather than what they say they'll do or you wish they'd do, or that your users are not like yourself. To my fellow instructors and me belongs the job of channeling that power, guiding it, shaping it for the benefit of humanity without squashing that spark. I found it humbling, and if you've read my columns, you know that I don't humble easily.

My most moving experience was accompanying the students to the Statue of Liberty and Ellis Island, in New York Harbor. About one-third of all Americans, including myself, have an ancestor who



The Members of Team Hawk Stand in Front of the Statue of Liberty

passed through the immigration station here, in the late 19th or early 20th centuries. It's been renovated now, scrubbed way too clean. But you can still feel the ghosts as the sun sets and the city skyline lights up across the water, deepening the darkness on the island.

New York still welcomes immigrants who have the right skills. "Today over 40 percent of New Yorkers are foreign born," said Mayor Michael Bloomberg at the awards presentation. "So when you finish your schooling, come join us. Bring your brains; we need them, and this is a great place to use them."

On the boat I met the members of Team Hawk—Choman, Kosar and Enji—representing Iraq. Their entry was a system for quickly registering arrivals in a refugee camp. They're a proud team from a proud country. "We're the world's oldest civilization," Choman told me. "Mesopotamia, between the Tigris and Euphrates rivers. But somehow we lost our way, and we have to get it back. That's what we're doing here."

The Statue of Liberty is a great place for that sort of thought. She represents to the world what's best about America, a clear symbol of

everything good that we have ever been or meant. The boat sails in front of her copper verdigris face and her eyes follow you, even though you know they can't. If you can keep from misting up at that sight, you're stronger than I am. Or maybe weaker, I'm not sure.

"Lift your head up," her torch says. Life looks very different from that angle. Try it some time.

Enji was in heaven. "I've dreamed to see that statue since I was a child, and now I can't believe I'm here. She's so beautiful. I love you, My Lady," and I heard the capital letters in her voice. I'm glad I could show it to her, to all of them, and glad they could show it to me, too. ■

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.

Richard **Campbell**

Carl **Franklin**

We Are Smart Developers

.NET Rocks!, the leading Internet audio talk show for .NET developers, recognizes the importance of using the best tools for the job.

GrapeCity PowerTools delivers by providing award-winning Reporting and Spreadsheet components.

Download your FREE trials.

Smarter Components for
SMARTER DEVELOPERS

GrapeCity PowerTools

www.GCPowerTools.com
GvTv.GCPowerTools.com



Syncfusion: Essential Studio 2011 Volume 3

Volume 3 is a turning point in many ways. Release to release, we've continually advanced our business component offering. That still holds true, but what marks the difference in this release is our venture into HTML 5 for web and mobile, our commitment to Windows Phone 7, and our RDL option for reporting.



HTML 5 Magic

The latest incarnation of HTML is set to revolutionize the web experience. For .NET developers, the arena has to be entered early. With five new HTML-5 based controls already developed, your entry will be easy, and will be done with the skill set you currently have. When everything is changing, you haven't time to waste.



RDL Reporting

Digital information often outlives the applications that access it. And while industry-standard SQL and OLAP stores give life to data, their reports are typically doomed to proprietary formats. With our 100% RDL reporting, you can freely interoperate with other products—knowing that your reports, like your data, will persist.



More Business Tools

We've added three substantial controls to our business toolbox. Each can change the way an organization operates: a spreadsheet control that allows you to see and edit .xls and .xlsx files without Excel; a PDF viewer for reading PDF files without relying on Adobe; and an interactive Gantt control for project management—all amounting to cost savings and added functionality.

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

 **Syncfusion®**
Deliver innovation with ease®