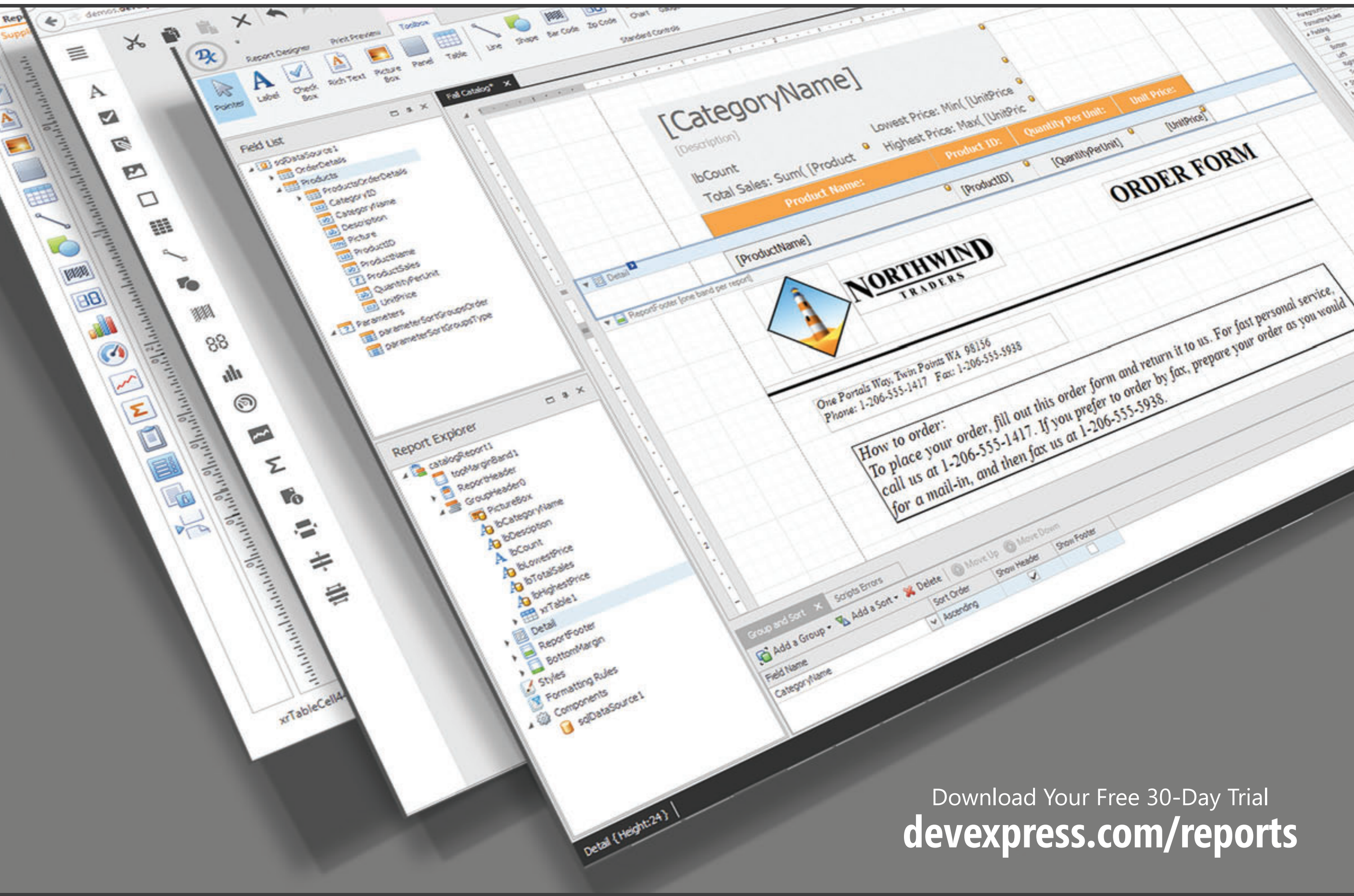
THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2015 VOL 30 NO 7

msdn magazine

The Easier Way to Create Reports

A Report Platform optimized for WPF, ASP.NET and Windows Forms developers.


Report Designer
Field List
Report Explorer
[CategoryName]
ORDER FORM
NORTHWIND TRADERS
How to order:
To place your order, fill out this order form and return it to us. For fast personal service, call us at 1-206-555-1417. If you prefer to order by fax, prepare your order as you would for a mail-in, and then fax us at 1-206-555-5938.
Group and Sort
Detail { Height:24 }

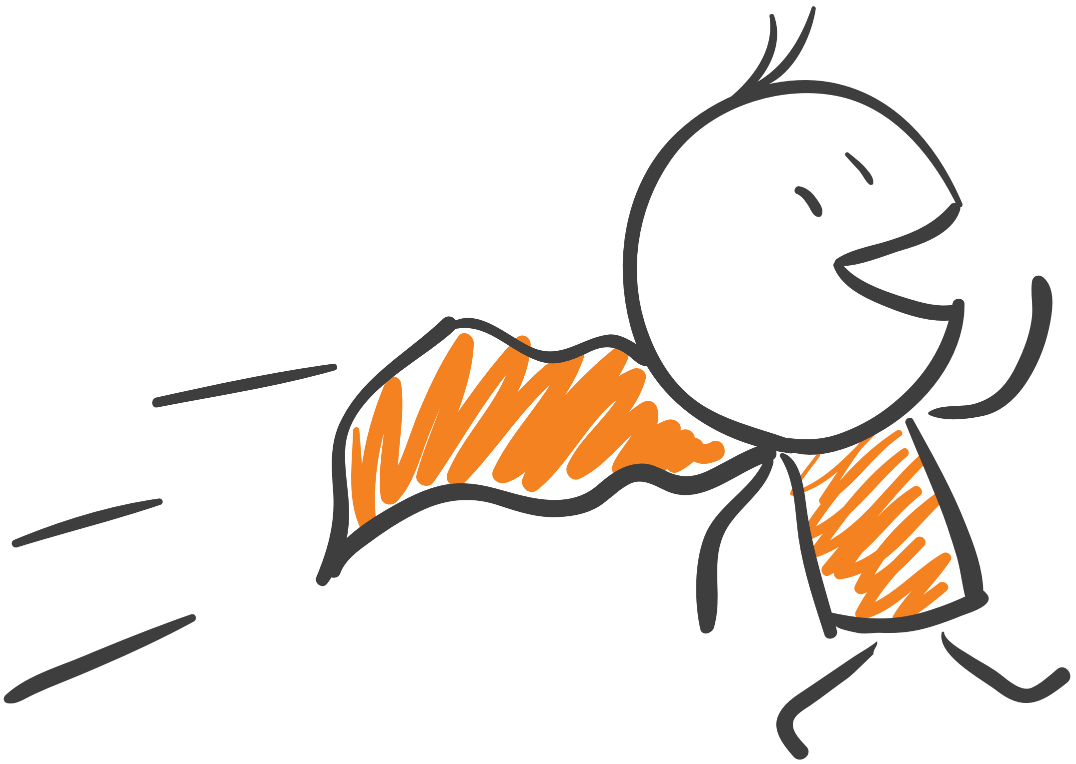
Download Your Free 30-Day Trial
devexpress.com/reports

All trademarks or registered trademarks are property of their respective owners.

DevExpress®

Unleash the UI Superhero in You

With DevExpress tools, you'll deliver amazing user-experiences for Windows®, the Web and Your Mobile World



Experience the DevExpress difference today.
Download your free 30-day trial.
devexpress.com/try

DevExpress® WIN ASP MVC WPF

THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2015 VOL 30 NO 7

msdn magazine

Microsoft

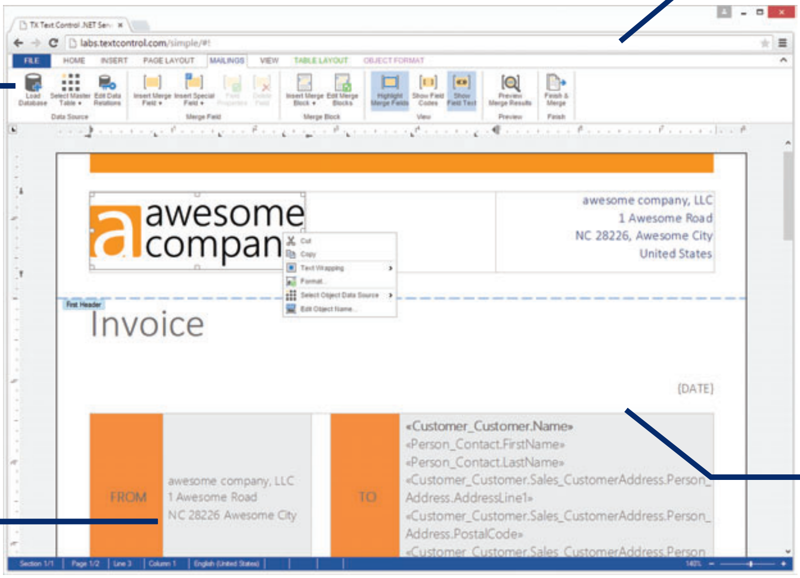
TEXT CONTROL

Reporting!

Combine powerful reporting with easy-to-use word processing

The **Text Control Reporting Framework** combines powerful reporting features with an easy-to-use, MS Word compatible word processor. Users create documents and templates using ordinary MS Word skills. TX Text Control is completely independent from MS Word or any other third-party application and can be completely integrated into your business application.

ASP.NET ▪ Windows Forms ▪ WPF



awesome company

awesome company, LLC
1 Awesome Road
NC 28226, Awesome City
United States

Invoice

[DATE]

FROM

awesome company, LLC
1 Awesome Road
NC 28226 Awesome City

TO

«Customer_Customer.Name»
«Person_Contact.FirstName»
«Person_Contact.LastName»
«Customer_Customer.Sales_CustomerAddress.Person_Address.AddressLine1»
«Customer_Customer.Sales_CustomerAddress.Person_Address.PostalCode»
«Customer_Customer.Sales_CustomerAddress.Person

Database support for ADO.NET, ODBC, DataSet, DataTable and all IEnumerable business objects

Cross-browser, cross-platform document and template editing

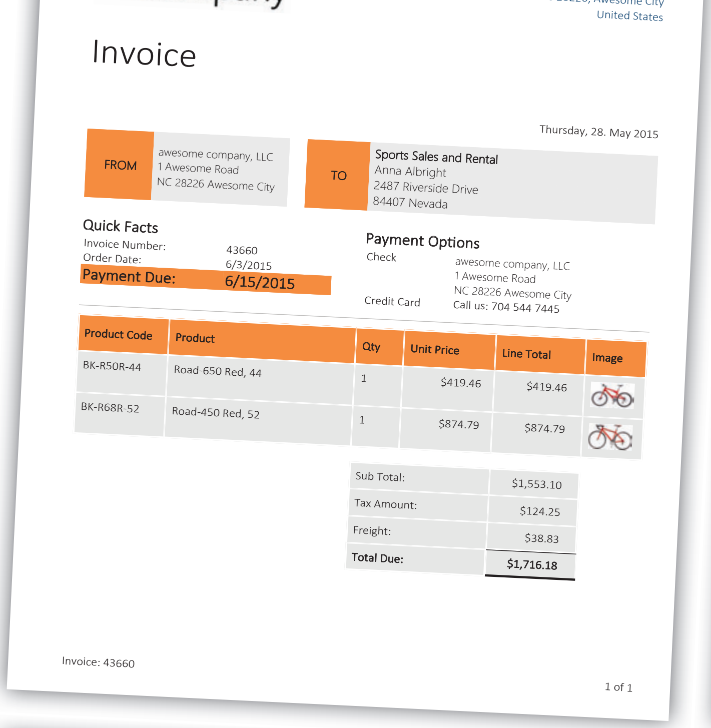
Create Adobe PDF and PDF/A documents

MS Word compatible templates and MS Word inspired UI

Visual Studio Partner

HTML

© 2015 Text Control GmbH. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective owners.



28226, Awesome City
United States

Invoice

Thursday, 28. May 2015

FROM	awesome company, LLC 1 Awesome Road NC 28226 Awesome City	TO	Sports Sales and Rental Anna Albright 2487 Riverside Drive 84407 Nevada

Quick Facts

Invoice Number:	43660
Order Date:	6/3/2015
Payment Due:	6/15/2015

Payment Options

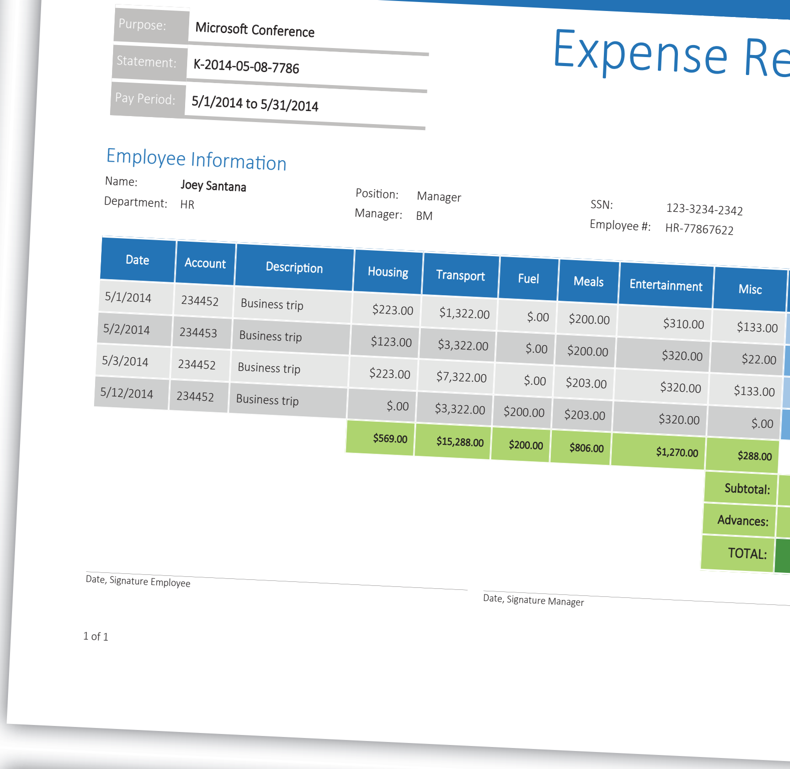
Check	awesome company, LLC 1 Awesome Road NC 28226 Awesome City
Credit Card	Call us: 704 544 7445

Product Code	Product	Qty	Unit Price	Line Total	Image
BK-R50R-44	Road-650 Red, 44	1	$419.46	$419.46	
BK-R68R-52	Road-450 Red, 52	1	$874.79	$874.79	

Sub Total:	$1,553.10
Tax Amount:	$124.25
Freight:	$38.83
Total Due:	$1,716.18

Invoice: 43660

1 of 1



Purpose:	Microsoft Conference
Statement:	K-2014-05-08-7786
Pay Period:	5/1/2014 to 5/31/2014

Expense Re

Employee Information

Name:	Joey Santana	Position:	Manager	SSN:	123-3234-2342
Department:	HR	Manager:	BM	Employee #:	HR-77867622

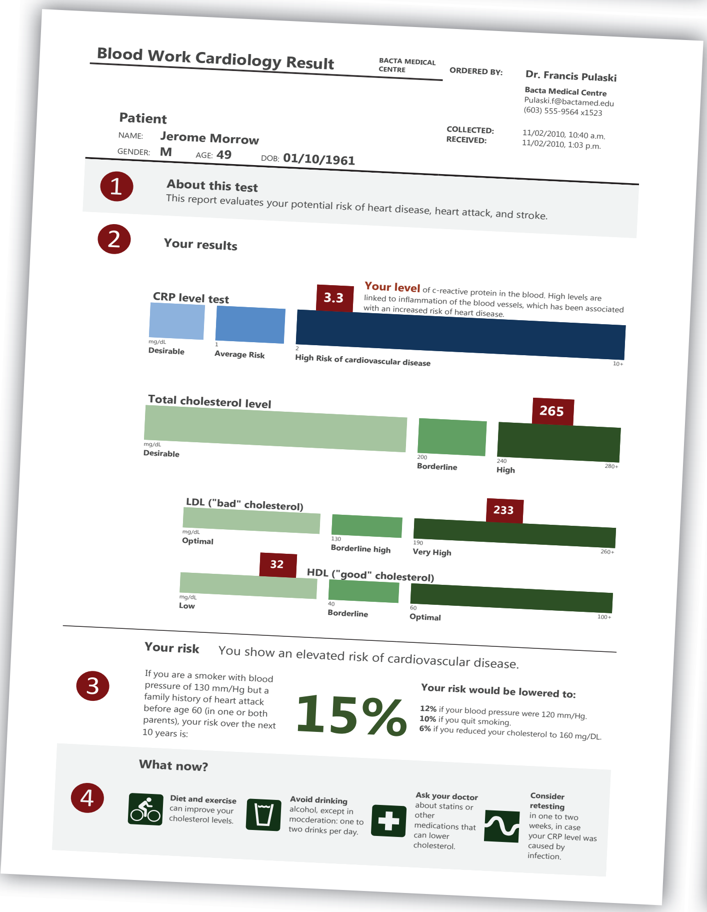
Date	Account	Description	Housing	Transport	Fuel	Meals	Entertainment	Misc
5/1/2014	234452	Business trip	$223.00	$1,322.00	$.00	$200.00	$310.00	$133.00
5/2/2014	234453	Business trip	$123.00	$3,322.00	$.00	$200.00	$320.00	$22.00
5/3/2014	234452	Business trip	$223.00	$7,322.00	$.00	$203.00	$320.00	$133.00
5/12/2014	234452	Business trip	$.00	$3,322.00	$200.00	$203.00	$320.00	$.00
			$569.00	$15,288.00	$200.00	$806.00	$1,270.00	$288.00

Subtotal:
Advances:
TOTAL:

Date, Signature Employee

Date, Signature Manager

1 of 1



Blood Work Cardiology Result

BACTA MEDICAL CENTRE

ORDERED BY: **Dr. Francis Pulaski**
Bacta Medical Centre
Pulaski.f@bactamed.edu
(603) 555-9564 x1523

Patient

NAME: **Jerome Morrow**
GENDER: **M** AGE: **49** DOB: **01/10/1961**

COLLECTED: 11/02/2010, 10:40 a.m.
RECEIVED: 11/02/2010, 1:03 p.m.

1 **About this test**
This report evaluates your potential risk of heart disease, heart attack, and stroke.

2 **Your results**

Your risk You show an elevated risk of cardiovascular disease.

3 If you are a smoker with blood pressure of 130 mm/Hg but a family history of heart attack before age 60 (in one or both parents), your risk over the next 10 years is:

15%

Your risk would be lowered to:

12% if your blood pressure were 120 mm/Hg.
10% if you quit smoking.
6% if you reduced your cholesterol to 160 mg/DL.

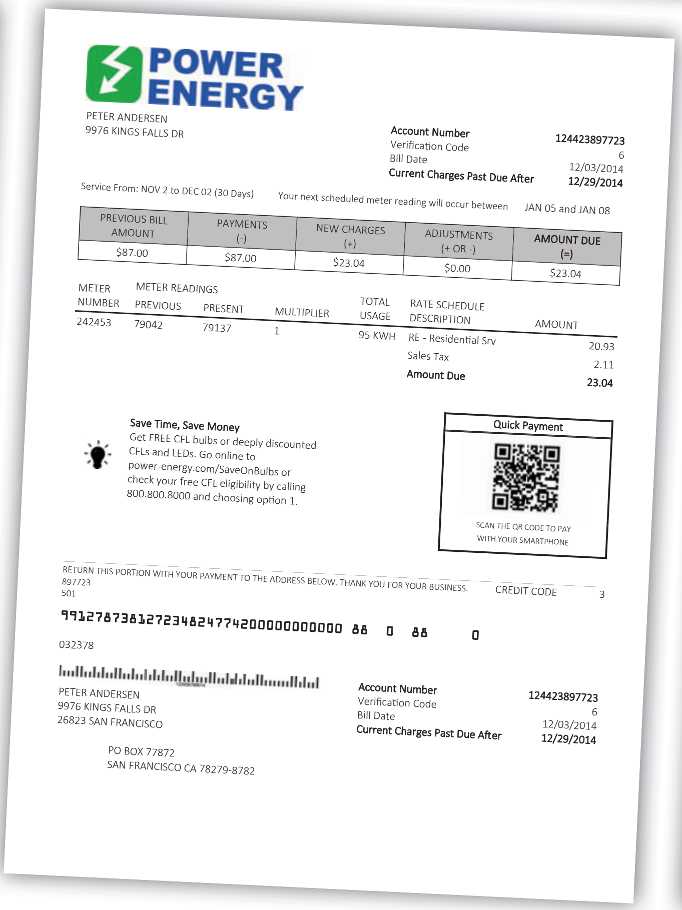
What now?

4 **Diet and exercise** can improve your cholesterol levels.

Avoid drinking alcohol, except in mocderation: one to two drinks per day.

Ask your doctor about statins or other medications that can lower cholesterol.

Consider retesting in one to two weeks, in case your CRP level was caused by infection.



POWER ENERGY

PETER ANDERSEN
9976 KINGS FALLS DR

Account Number	124423897723
Verification Code	6
Bill Date	12/03/2014
Current Charges Past Due After	12/29/2014

Service From: NOV 2 to DEC 02 (30 Days) Your next scheduled meter reading will occur between JAN 05 and JAN 08

PREVIOUS BILL AMOUNT	PAYMENTS (-)	NEW CHARGES (+)	ADJUSTMENTS (+ OR -)	AMOUNT DUE (=)
$87.00	$87.00	$23.04	$0.00	$23.04

METER NUMBER	METER READINGS PREVIOUS	PRESENT	MULTIPLIER	TOTAL USAGE	RATE SCHEDULE DESCRIPTION	AMOUNT
242453	79042	79137	1	95 KWH	RE - Residential Srv	20.93
					Sales Tax	2.11
					Amount Due	23.04

Save Time, Save Money
Get FREE CFL bulbs or deeply discounted CFLs and LEDs. Go online to power-energy.com/SaveOnBulbs or check your free CFL eligibility by calling 800.800.8000 and choosing option 1.

Quick Payment

SCAN THE QR CODE TO PAY WITH YOUR SMARTPHONE

RETURN THIS PORTION WITH YOUR PAYMENT TO THE ADDRESS BELOW. THANK YOU FOR YOUR BUSINESS. CREDIT CODE 3
897723
501

9912787381272348247742000000000000 88 0 88 0

032378

PETER ANDERSEN
9976 KINGS FALLS DR
26823 SAN FRANCISCO

Account Number	124423897723
Verification Code	6
Bill Date	12/03/2014
Current Charges Past Due After	12/29/2014

PO BOX 77872
SAN FRANCISCO CA 78279-8782

Sports Sales
Ms. Anna Albrigh
2487 Riverside D
84407 Nevada

Order Date: 00/01

SH

Cross-Country
Mr. Paul Alcorn
8291 Crossbow Way
38103 Memphis

Order Date: 00/01

Live demos and 30-day trial version download at:

www.textcontrol.com/reporting

X12

TEXT CONTROL

msdn magazine

JULY 2015 VOLUME 30 NUMBER 7

General Manager Jeff Sandquist
Director Keith Boyd
Editorial Director Mohammad Al-Sabt *mmeditor@microsoft.com*
Site Manager Kent Sharkey
Editorial Director, Enterprise Computing Group Scott Bekker
Editor in Chief Michael Desmond
Features Editor Lafe Low
Features Editor Sharon Terdeman
Group Managing Editor Wendy Hernandez
Senior Contributing Editor Dr. James McCaffrey
Contributing Editors Rachel Appel, Dino Esposito, Kenny Kerr, Julie Lerman, Ted Neward, David S. Platt, Bruno Terkaly, Ricardo Villalobos
Vice President, Art and Brand Design Scott Shultz
Art Director Joshua Gould

ENTERPRISE COMPUTINGGROUP

President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

ART STAFF
Creative Director **Jeffrey Langkau**
Associate Creative Director **Scott Rovin**
Senior Art Director **Deirdre Hoffman**
Art Director **Michele Singh**
Assistant Art Director **Dragutin Cvijanovic**
Graphic Designer **Erin Horlacher**
Senior Graphic Designer **Alan Tao**
Senior Web Designer **Martin Peace**

PRODUCTION STAFF
Director, Print Production **David Seymour**
Print Production Coordinator **Anna Lyn Bayaua**

ADVERTISING AND SALES
Chief Revenue Officer **Dan LaBianca**
Regional Sales Manager **Christopher Kourtoglou**
Account Executive **Caroline Stover**
Advertising Sales Associate **Tanya Egenolf**

ONLINE/DIGITAL MEDIA
Vice President, Digital Strategy **Becky Nagel**
Senior Site Producer, News **Kurt Mackie**
Senior Site Producer **Gladys Rama**
Site Producer **Chris Paoli**
Site Producer, News **David Ramel**
Senior Site Administrator **Shane Lee**
Site Administrator **Biswarup Bhattacharjee**
Senior Front-End Developer **Rodrigo Munoz**
Junior Front-End Developer **Anya Smolinski**
Executive Producer, New Media **Michael Domingo**
Office Manager & Site Assoc. **James Bowling**

LEAD SERVICES
Vice President, Lead Services **Michele Imgrund**
Senior Director, Audience Development & Data Procurement **Annette Levee**
Director, Audience Development & Lead Generation Marketing **Irene Fincher**
Director, Client Services & Webinar Production **Tracy Cook**
Director, Lead Generation Marketing **Eric Yoshizuru**
Director, Custom Assets & Client Services **Mallory Bundy**
Editorial Director, Custom Content **Lee Pender**
Senior Program Manager, Client Services & Webinar Production **Chris Flack**
Project Manager, Lead Generation Marketing **Mahal Ramos**
Coordinator, Lead Generation Marketing **Obum Ukabam**

MARKETING
Chief Marketing Officer **Carmel McDonagh**
Vice President, Marketing **Emily Jacobs**
Senior Manager, Marketing **Christopher Morales**

ENTERPRISE COMPUTING GROUP EVENTS
Senior Director, Events **Brent Sutton**
Senior Director, Operations **Sara Ross**
Director, Event Marketing **Merikay Marzoni**
Events Sponsorship Sales **Danna Vedder**
Senior Manager, Events **Danielle Potts**
Coordinator, Event Marketing **Michelle Cheng**
Coordinator, Event Marketing **Chantelle Wallace**

1105 MEDIA INC

Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Senior Vice President & Chief Financial Officer
Richard Vitale

Executive Vice President
Michael J. Valenti

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

Chairman of the Board
Jeffrey S. Klein

ID STATEMENT 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, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. 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, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 www.1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@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, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jlong@meritdirect.com; Web: www.meritdirect.com/1105

Reaching the Staff
Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at Redmondmag.com.
E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618
Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311
The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA A Century Moving Business Forward

WPA Western Publications Association

LEADTOOLS® V19

THE WORLD LEADER IN IMAGING SDKs

Forms Recognition and Processing

Recognize and process structured and unstructured forms including invoices, passports, driver's licenses and checks

- ◇ Recognize and extract form fields regardless of image resolution, scale, and other form generation characteristics (OCR, OMR, ICR, 1D & 2D Barcodes)
- ◇ Advanced form alignment algorithm compensates for non-linear deformations introduced by different scanners, printers and resolutions
- ◇ Automatically detect and correct page orientation and skew angle
- ◇ Recognize both vertical and horizontal text from the same document
- ◇ Comprehensive confidence reporting for each form field type
- ◇ World-class accuracy and speed resulting in significant savings of time and resources

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Who We Are

In May we completed our biannual survey of *MSDN Magazine* subscribers. This is the third such survey we've conducted since 2011, and the results provide a valuable picture of the evolving *MSDN Magazine* readership.

For instance, did you know that *MSDN Magazine* readers are getting younger? It's true. This year, 31 percent of respondents report having 20 or more years of experience in the industry, versus 37.7 percent in 2013 and 36 percent in 2011. At the same time, the percentage of subscribers with less than 10 years of experience has risen, from 22 percent in 2011 and 2013 to 26.4 percent this year.

In terms of language use, C# remains king. It was identified as the primary programming language of 70.1 percent of respondents, up from 65.5 percent in 2013.

We also asked developers what initiatives are important to them, and found two areas that produced the biggest gains. Agile development is up sharply since 2011, from 53 percent of respondents to 64.4 percent this year. Cloud development is also up big, earning a nod from 39.9 percent of respondents, up from 29 percent in 2011 and 36.6 percent in 2013. Another significant gainer: application lifecycle management, which was identified by 41.7 percent of respondents this year, versus 35 percent in 2011 and 40.2 percent in 2013. Two other initiatives—security and mobile/wireless development—also saw gains.

A pair of initiatives lost the most ground over the last four years. The Rich Client Applications - BI/Database/Information Worker category fell from a 33 percent response rate in 2011 to just 19.2 percent in 2015. GUI design and usability development also tailed off, from 62 percent in 2011 to 49.8 percent in 2015.

In terms of language use, C# remains king. It was identified as the primary programming language of 70.1 percent of respondents, up from 65.5 percent in 2013. C++ rebounded this year to capture the loyalty of nearly 10 percent of respondents, from just 6.1 percent in 2013. But Visual Basic usage shows persistent decline. The percentage of readers using the language has dropped over each of the last three surveys, from 17 percent to 12.3 percent to just 8.6 percent in 2015. The only other languages to gain more than 2 percent of responses were Java and JavaScript, with both at just a shade less than 3 percent.

When looking at Microsoft technologies that developers either have or plan to invest in within the next 12 months, the clear trend is toward cloud adoption. Microsoft Azure was cited by 36 percent of respondents, up from 28.4 percent in 2013, while Azure SQL Database was cited by 23.4 percent this year, from 18.8 percent two years earlier. Not coincidentally, planned usage of SQL Server is dropping, from 82 percent in 2011 to 78.5 percent in 2013, to 73.7 percent this year.

2015 is very much a year in transition. Usage of technologies like SharePoint, ASP.NET and especially Silverlight waned, while uptake of Windows Runtime remains steady. Of interest: Survey respondents report declining uptake of Windows Phone, from 21.3 percent two years ago to 16.1 percent in 2015. That will be a figure to watch, especially as Microsoft's Universal Windows apps strategy begins to flatten barriers to entry in the Windows Phone space.

This year, we decided to ask our readers where they go for help and guidance with development projects. No surprise, *MSDN Magazine* subscribers favored MSDN.com as a primary source of insight, with three-quarters of all respondents listing it. The StackOverflow site was a close second, identified by 64 percent of respondents. The next-most-cited sources were Google (19 percent), W3Schools (17 percent) and Pluralsight (14 percent).

Over the past four years we've seen Microsoft adapt to a world defined by mobile, cloud and Web development. I look forward to seeing how our readership continues to adapt in the years to come.

[illegible]

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

© 2015 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.

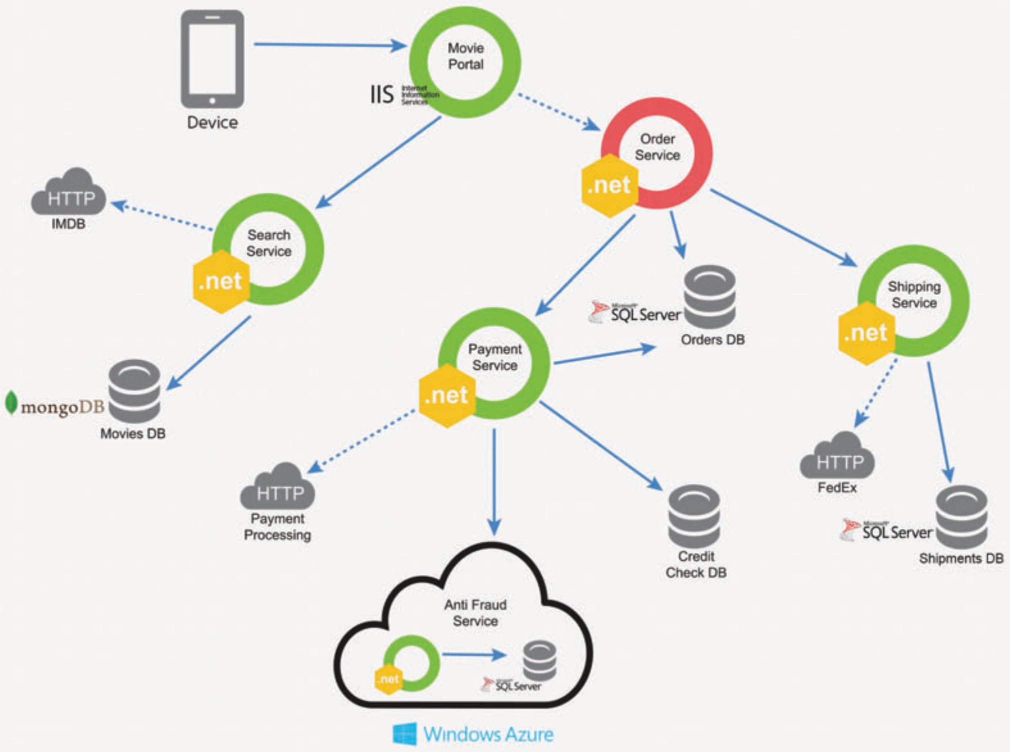
A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

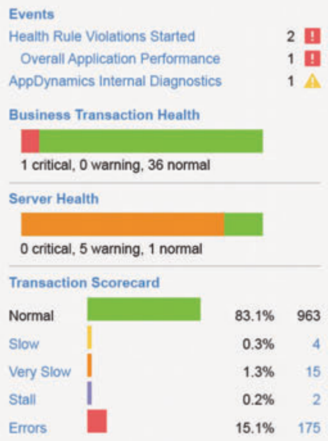
MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific 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.

APPDYNAMICS

There's nothing about .NET that AppDynamics doesn't see.

AppDynamics gives you the visibility to take command of your .NET application's performance, no matter how complicated or distributed your environment is.





Events

Health Rule Violations Started	2	❗
Overall Application Performance	1	❗
AppDynamics Internal Diagnostics	1	⚠

Business Transaction Health

1 critical, 0 warning, 36 normal

Server Health

0 critical, 5 warning, 1 normal

Transaction Scorecard

Normal	83.1%	963
Slow	0.3%	4
Very Slow	1.3%	15
Stall	0.2%	2
Errors	15.1%	175

When your business runs on .NET and SQL Server, count on AppDynamics to give you the complete visibility you need to be sure they are delivering the performance and business results you need — no matter how complex, distributed or asynchronous your environment, live 'in production' or during development.

See every line of code. Get a complete view of your environment with deep code diagnostics and auto-discovery. Understand performance trends with dynamic baselining. And drastically reduce time to root cause and remediation.

See why the world's largest .NET deployments rely on the AppDynamics Application Intelligence Platform. Sign up for a FREE trial today at **www.appdynamics.com/Microsoft**.

appdynamics.com

Journey to Windows 10

In 2001 I received my first Microsoft "Ship It" awards for Internet Explorer (IE) 6 and Windows XP. The initial reaction to Windows XP is humorous in hindsight. Pundits were aghast at the new "Aero" UI and predicted doom for the first Windows release featuring a combined Windows 9x and NT kernel. For a few months, sales were tepid, and Microsofties everywhere held their breath. Then there was a collective exhale as consumers and businesses made Windows XP a smash hit.

Sure, there were problems along the way—the number of exploits in Windows XP forced the entire company to stop and spend months on end doing a "security push." To this day, there are copies of David LeBlanc's "Writing Secure Code" on shelves all over Microsoft. Windows XP might have been chock full of security holes, but it was subsequently hardened, improved upon, and to this day remains one of the most beloved products in Microsoft history. Microsoft began the Windows Vista project with its mojo intact.

Even on machines without touch, Windows 8.1 is easy to use and navigate.

As Windows Vista development began, the IE team was famously dismantled, in favor of a new team tasked to build a modern, connected-client platform. The "Avalon" team (of which I was a part) built the managed application platform known as Windows Presentation Foundation, or WPF. Throughout the myriad missteps of the Vista project, my conviction about WPF as a platform that could propel Microsoft forward was resolute.

Alas, the travails of Windows Vista are well documented. We finally shipped five and a half years later, and while there was legitimate excitement for WPF and other new technologies, Windows Vista failed to ignite the virtuous cycle that propelled the PC industry like past releases. It was the first legitimate dud in a long time.

Enter Steven Sinofsky.

Much like the "Star Trek" movie franchise, Windows was enjoying success with every other release. Sinofsky aimed to stop that by infusing a new confidence and culture in the Windows team, and under his stewardship he produced the widely respected—and even loved—Windows 7. But also staying true to the "Star Trek" analogy, that same leadership team, riding a high similar to that of the earlier Windows management team that successfully fused the NT and 9x kernels in only 18 months, went for a moon shot—and produced Windows 8.

Like Windows Vista, Windows 8 struggled out of the gate. However, the product itself improved markedly with Windows 8.1. In fact, any early user of Windows 8 who was turned off by the new "modern" UI should give Windows 8.1 a look. Even on machines without touch, Windows 8.1 is easy to use and navigate. The new app platform has caught on, with amazing apps and games added to the Windows Store every day. It might have had a slow start, but Windows 8.1 has Windows back on the right path.

And boy, did Terry Myerson, Michael Fortin, Joe Belfiore and the rest of the Microsoft Operating System Group leadership team seize the moment with Windows 10. I've been using an early beta for several months now and I love it. All input modalities, from touch, to voice, to mouse and keyboard, feel completely natural and intuitive. The "modern" concepts introduced in Windows 8 are now seamlessly integrated into the OS. The fit and finish—even on my early build—is amazing. Stability, with very few exceptions, has been stellar. Many of you are experiencing Windows 10 for the first time after Build and Ignite and I suspect your reaction is much the same as mine: Wow. Windows is back. And while I might not be a member of the team anymore, I couldn't be prouder.

If my "Star Trek" analogy holds true, we'll say that Windows 2000 represents "Star Trek: the Motion Picture"—groundbreaking, but boring. Windows XP is analogous to the glorious "The Wrath of Khan." Windows Vista maps to "The Search for Spock," which might be retitled "The Search for Relevance." Windows 7 is "The Voyage Home," a release where we came home to the fundamentals of Windows. Hopefully Windows 8 isn't the "The Final Frontier," but it did receive a similar reaction from critics.

What does this mean for Windows 10? Perhaps it's apt that the last movie with the original "Star Trek" crew was titled "The Undiscovered Country." It wasn't necessarily my favorite "Star Trek" movie, but it was solid, nostalgic, and packed a few surprises. But I actually think you can expect a lot more from Windows 10. In fact, with all that has changed in the latest version of Windows, it may be best to think of it as a full-blown, J.J. Abrams-style reboot. It's that good. ■

KEITH BOYD *manages the developer documentation team in the Cloud and Enterprise (C&E) division at Microsoft. In that capacity, he oversees editorial strategy for* MSDN Magazine. *Prior to arriving in C&E in 2013, he held the same role in Windows. When he's not at work, he's enjoying time with his family, including his wife, Leslie, and three sons, Aiden, Riley and Drew.*

/n software

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 …

Microsoft .NET | Microsoft BizTalk Server | Microsoft SQL Server | Windows Mobile | activex | Microsoft SharePoint 2010 | CodeGear DELPHI | CodeGear C++BUILDER | Java | php

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

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

Why It's Hard to Talk About Tech

The hardest part about developing software could be talking about it. Every day, developers must articulate deep technical detail and broad logical concepts, often to people with only a passing understanding of what they are talking about.

I see people struggle with this all the time and I don't think it's because they're poor communicators or have bad ideas. It's because talking about something as abstract as code is extremely difficult. This wouldn't be such a problem if communications were a rare occurrence in software engineering, but in fact the opposite is true. Effective communication is at the heart of every successful, team-based software development effort.

Even if you're a freelance programmer, trying to understand and deliver what a customer wants can be strenuous work. The challenge gets even tougher on large projects with many developers, where each team member is working on different pieces of a puzzle that must fit together.

Maybe talking about tech will always be difficult, or maybe as the relatively young discipline of software development matures, we'll see more effective methods of communication develop.

Showing someone some code, or even pseudo code, can be a great way to get your ideas across, but sometimes the code isn't available, or it's too large to be immediately grasped by others during a conversation. As a result, we're often relegated to hand waving and lofty descriptions.

Another pitfall is when developers assume the people they're talking to are familiar with the jargon and acronyms they fit into their explanations, or that others have the prerequisite knowledge of the helper functions being referenced. If the other party has a shaky grasp of these concepts, it will slow down—if not stop completely—the journey to understanding. As soon as a person hears an unfamiliar acronym, much of the following dialog is lost to them as they struggle for context.

More broadly, a successful dialog about programming requires that both parties hold a similar blueprint in their minds at the start of the conversation. This blueprint can be the context of the code you're writing or even the architecture of the whole project. While this bit is normally explained out of necessity, it's not always explained very well—UML diagrams are normally too inconvenient and often there simply isn't time to review all of the objects and data structures in detail. The upshot: Teams often fail to meet requirements to drive informed decisions, yet we all continue to do it anyway.

Communication is a tough challenge and one with which I personally struggle. But it's a weakness that, if acknowledged, can be worked on. As developers, we must remind ourselves that others may not be following our train of thought as closely as we believe. On the flip side, we must be willing to let others know when we're struggling to understand a concept before the conversation moves on to the next topic.

Given all of these difficulties, I find it to be a triumph of software engineering that a product like Windows even works at all, given the thousands of engineers who have worked on it over the years.

Maybe talking about tech will always be difficult, or maybe as the relatively young discipline of software development matures, we'll see more effective methods of communication develop. Perhaps the solution lies in technology itself, with better tools for collaboration and more focused and evolved programming languages. If the programming languages of the future are designed with clearer and intuitive syntax, and do away with the verbose baggage carried over from primitive languages, we may see code that we can more quickly read and whose meaning is more easily absorbed. Maybe part of the solution lies in newer, better programming paradigms, which will allow for better analogy-driven explanations than even object-oriented programming can provide now.

One thing is clear, whoever creates and adopts these solutions will gain a significant edge over those who continue to wander in the sandstorm of acronyms and hand waving that characterize software engineering today. ■

RYDER DONAHUE *is a software developer engineer at Microsoft. Originally from the Hawaiian Islands, he now resides in Redmond, Wash., with his fiancée and their cat, Marbles.*

ComponentSource® /update/2015/07

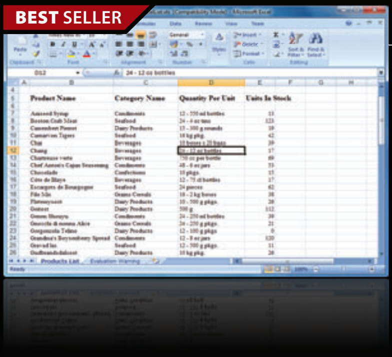
Experience the brand new look at **www.componentsource.com**

Help & Manual Professional | from $583.10

ec software

Help and documentation for .NET and mobile applications.

- Powerful features in an easy, accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to responsive HTML, CHM, PDF, MS Word, ePUB, Kindle or print
- Styles and Templates give you full design control

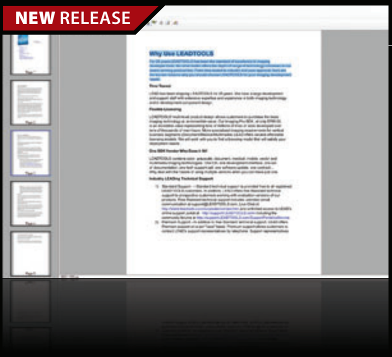


Aspose.Total for .NET | from $2,449.02

ASPOSE Your File Format APIs

Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR and OneNote files alongside many more document management features in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types



LEADTOOLS PDF Pro SDK V19 | from $1,495.00 SRP

LEAD TECHNOLOGIES INCORPORATED

Add powerful PDF read, write, view & editing functionality to desktop, tablet & mobile applications.

- View PDF files as raster in .NET, WinRT and C/C++
- Merge, Split, Convert, Linearize and Distill PDF files
- Read and Extract text, hyperlinks, metadata and more from PDF
- Create the smallest file possible utilizing advanced PDF Optimizer
- Convert PDF to an image format such as JPEG or TIFF

GrapeCity ComponentOne Studio 2015 V1 | from $1,315.60

ComponentOne Studio

.NET Controls for Serious Application Development

- Solutions for Data Visualization, Data Management, Reporting and Business Intelligence
- Hundreds of .NET UI controls for all Visual Studio platforms
- Built-in themes and an array of designers for custom styling
- Adaptive, mobile-friendly and touch-enabled controls

© 1996-2015 ComponentSource. All Rights Reserved. All prices correct at the time of press. Online prices may vary from those shown due to daily fluctuations & online discounts.

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

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

European Headquarters
ComponentSource
2 New Century Place
East Street
Reading, Berkshire
RG1 4ET
United Kingdom

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

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

VISA DISCOVER

GSA Schedule Contract GS-35F-0188R

CQRS and Message-Based Applications

At the end of the day, Command and Query Responsibility Segregation (CQRS) is software design that separates the code that alters state and the code that just reads state. That separation can be logical and based on different layers. It can also be physical and involve distinct tiers. There's no manifesto or trendy philosophy behind CQRS. The only driver is its simplicity of design. A simplified design in these crazy days of overwhelming business complexity is the only safe way to ensure effectiveness, optimization and success.

My last column (msdn.microsoft.com/magazine/mt147237) offered a perspective of the CQRS approach that made it suitable for any type of application. The moment you consider using a CQRS architecture with distinct command and query stacks, you start thinking of ways to separately optimize each stack.

There are no longer model constraints that make certain operations risky, impractical or perhaps just too expensive. The vision of the system becomes a lot more task-oriented. More important, it happens as a natural process. Even some domain-driven design concepts such as aggregates stop looking so irksome. Even they find their natural place in the design. This is the power of a simplified design.

If you're now curious enough about CQRS to start searching for case studies and applications close to your business, you may find most references refer to application scenarios that use events and messages to model and implement business logic. While CQRS can happily pay the bills with far simpler applications—those you might otherwise label as plain CRUD apps—it definitely shines in situations with greater business complexity. From that, you can infer greater intricacy of business rules and high inclination to change.

Message-Based Architecture

While observing the real world, you'll see actions in process and events that result from those actions. Actions and events carry data and sometimes generate new data, and that's the point. It's just data. You don't necessarily need a full-fledged object model to support executing these actions. An object model can still be useful. As you'll see in a moment, though, it's just another possible option for organizing business logic.

Figure 1 **Defining the Base Message Class**

```
public class Message
{
  public DateTime TimeStamp { get; proteted set; }
  public string SagaId { get; protected set; }
}
public class Command : Message
{
  public string Name { get; protected set; }
}
public class Event : Message
{
  // Any properties that may help retrieving
  // and persisting events.
}
```

A message-based architecture is beneficial as it greatly simplifies managing complex, intricate and frequently changing business workflows. These types of workflows include dependencies on legacy code, external services and dynamically changing rules. However, building a message-based architecture would be nearly impossible outside the context of CQRS that keeps command and query stacks neatly separated. Therefore, you can use the following architecture for the sole command stack.

A message can be either a command or an event. In code, you'd usually define a base Message class and from that, define additional base classes for commands and events, as shown in **Figure 1**.

From a semantic perspective, commands and events are slightly different entities and serve different but related purposes. An event is nearly the same as in the Microsoft .NET Framework: a class that carries data and notifies you when something that has occurred. A command is an action performed against the back end that a user or some other system component requested. Events and commands follow rather standard naming conventions. Commands are imperative like SubmitOrderCommand, while events are in past tense such as OrderCreated.

Typically, clicking any interface element originates a command. Once the system receives the command, it originates a task. The task can be anything from a long-running stateful process, a single action or a stateless workflow. A common name for such a task is saga.

A task is mono-directional, proceeds from the presentation down through the middleware and likely ends up modifying the system and storage state. Commands don't usually return data to the presentation, except perhaps some quick form of feedback such as whether the operation completed successfully or the reasons it failed.

Explicit user actions aren't the only way to trigger commands. You can also place a command with autonomous services that asynchronously interact with the system. Think of a B2B scenario, such as shipping products, in which communication between partners occurs over an HTTP service.

Events in a Message-Based Architecture

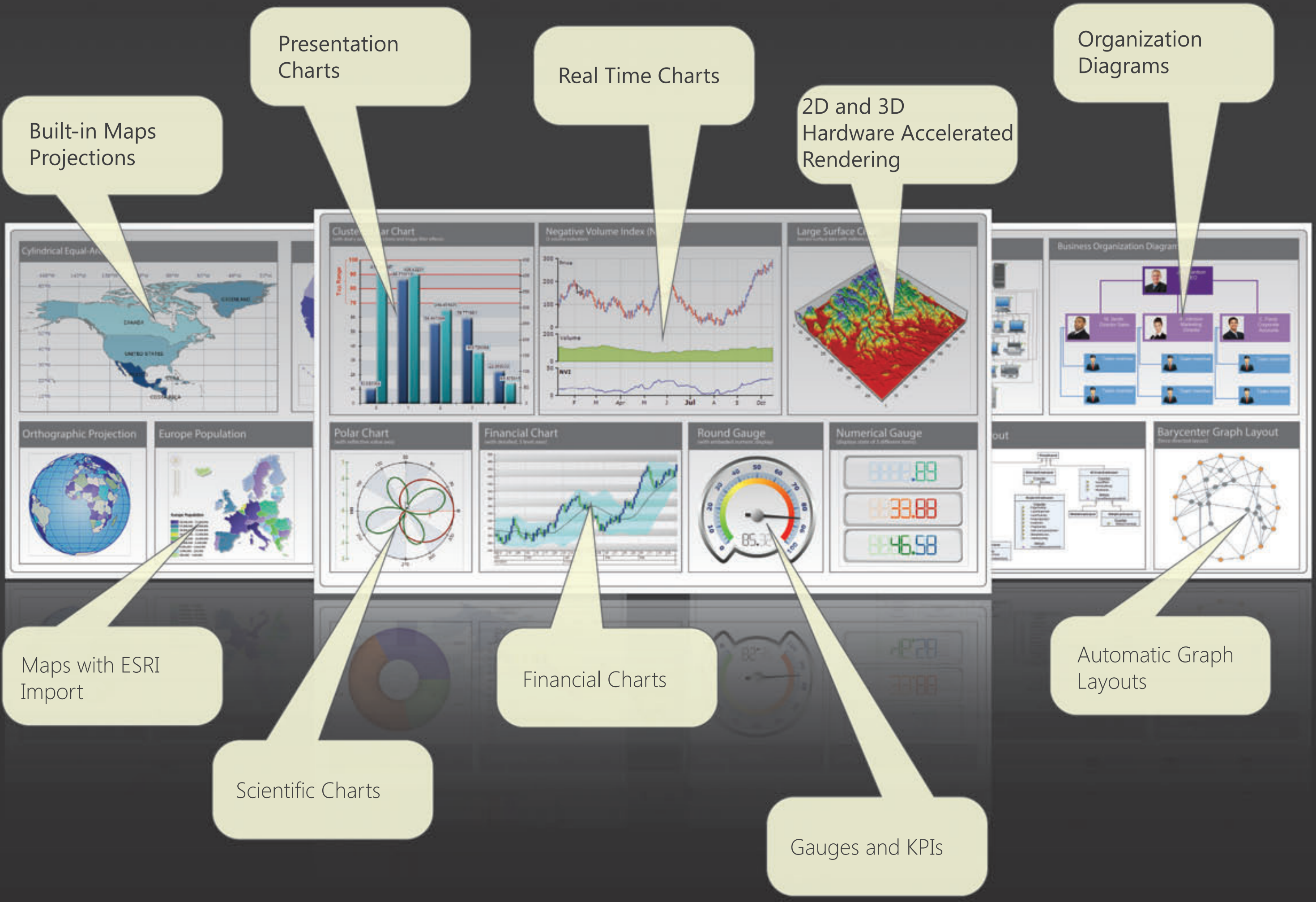
So commands originate tasks and tasks often consist of several steps that combine to form a workflow. Often when a given step executes, a results notification should pass to other components for additional work. The chain of sub-tasks triggered by a command can be long and complex. A message-based architecture is beneficial because it lets you model workflows in terms of individual

NEVRON www.nevron.com

Nevron Data Visualization

The leading data visualization components for desktop and web development in a single package.

NEW RELEASE 2015.1



solutions available for

Microsoft ASP.net | Microsoft Silverlight | WPF | WinForms | SharePoint | Microsoft SQl Server

Learn more at **www.nevron.com** today

www.nevron.com | email@nevron.com | +1 888-201-6088 (Toll free, USA and Canada)

Microsoft, .NET, ASP.NET, SharePoint, SQL Server and Visual Studio are registered trademarks of Microsoft Corporation in the United States and/or other countries.
Some Nevron components are only available for certain platforms. For details visit **www.nevron.com** or send and e-mail to **support@nevron.com**.

Visual Studio Partner

actions (triggered by commands) and events. By defining handler components for commands and subsequent events, you can model any complex business process.

More important, you can follow a working metaphor close to that of a classic flowchart. This greatly simplifies understanding the rules and streamlines communication with domain experts. Furthermore, the resulting workflow is broken into myriad smaller handlers, each performing a small step. Every step also places async commands and notifies other listeners of events.

One major benefit to this approach is the application logic is easily modifiable and extensible. All you do is write new parts and add them to the system, and you can do so with full certainty they won't affect existing code and existing workflows. To see why this is true and how it really works, I'll review some of the implementation details of message-based architecture, including a new infrastructural element—the bus.

Welcome to the Bus

To start out, I'll look at a handmade bus component. The core interface of a bus is summarized here:

```
public interface IBus
{
  void Send<T>(T command) where T : Command;
  void RaiseEvent<T>(T theEvent) where T : Event;
  void RegisterSaga<T>() where T : Saga;
  void RegisterHandler<T>();
}
```

Typically, the bus is a singleton. It receives requests to execute commands and event notifications. The bus doesn't actually do any

Figure 2 **Example of a Bus Class Implementation**

```
public class InMemoryBus : IBus
{
  private static IDictionary<Type, Type> RegisteredSagas =
    new Dictionary<Type, Type>();
  private static IList<Type> RegisteredHandlers =
    new List<Type>();
  private static IDictionary<string, Saga> RunningSagas =
    new Dictionary<string, Saga>();

  void IBus.RegisterSaga<T>()
  {
    var sagaType = typeof(T);
    var messageType = sagaType.GetInterfaces()
      .First(i => i.Name.StartsWith(typeof(IStartWith<>).Name))
      .GenericTypeArguments
      .First();
    RegisteredSagas.Add(messageType, sagaType);
  }
  void IBus.Send<T>(T message)
  {
    SendInternal(message);
  }
  void IBus.RegisterHandler<T>()
  {
    RegisteredHandlers.Add(typeof(T));
  }
  void IBus.RaiseEvent<T>(T theEvent)
  {
    EventStore.Save(theEvent);
    SendInternal(theEvent);
  }
  void SendInternal<T>(T message) where T : Message
  {
    // Step 1: Launch sagas that start with given message
    // Step 2: Deliver message to all already running sagas that
    // match the ID (message contains a saga ID)
    // Step 3: Deliver message to registered handlers
  }
}
```

concrete work. It just selects a registered component to process the command or handle the event. The bus holds a list of known business processes triggered by commands and events, or advanced by additional commands.

Processes that handle commands and related events are usually referred to as sagas. During initial bus configuration, you register handler and saga components. A handler is just a simpler type of saga and represents a one-off operation. When this operation is requested, it starts and ends without being chained to other events or by pushing other commands to the bus. **Figure 2** presents a possible bus class implementation that holds sagas and handlers references in memory.

When you send a command to the bus, it goes through a three-step process. First, the bus checks the list of registered sagas to see if there's any registered sagas configured to start upon receipt of that message. If so, a new saga component is instantiated, passed the message and added to the list of running sagas. Finally, the bus checks to see if there's any registered handler interested in the message.

An event passed to the bus is treated like a command and routed to registered listeners. If relevant to the business scenario, however, it may log an event to some event store. An event store is a plain append-only data store that tracks all events in a system. Using logged events varies quite a bit. You can log events for tracing purposes only or use that as the sole data source (event sourcing). You could even use it to track the history of a data entity while still using classic databases for saving the last-known entity state.

Writing a Saga Component

A saga is a component that declares the following information: a command or event that starts the business process associated with the saga, the list of commands the saga can handle, and the list of events in which the saga is interested. A saga class implements interfaces through which it declares the commands and events that are of interest. Interfaces like IStartWith and ICanHandle are defined as follows:

```
public interface IStartWith<T> where T : Message
{
  void Handle(T message);
}
public interface ICanHandle<T> where T : Message
{
  void Handle(T message);
}
```

Here's an example of the signature of a sample saga class:

```
public class CheckoutSaga : Saga<CheckoutSagaData>,
       IStartWith<StartCheckoutCommand>,
       ICanHandle<PaymentCompletedEvent>,
       ICanHandle<PaymentDeniedEvent>,
       ICanHandle<DeliveryRequestRefusedEvent>,
       ICanHandle<DeliveryRequestApprovedEvent>
{
  ...
}
```

In this case, the saga represents the checkout process of an online store. The saga starts when the user clicks the checkout button and the application layer pushes the Checkout command to the bus. The saga constructor generates a unique ID, which is necessary to handle concurrent instances of the same business process. You should be able to handle multiple concurrently running checkout sagas. The ID can be a GUID, a unique value sent with the command request or even the session ID.

For a saga, handling a command or event consists of having the Handle method on the ICanHandle or IStartWith interfaces invoked

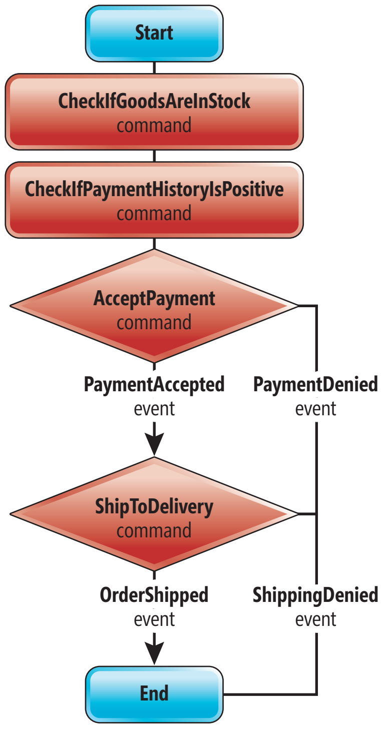


Figure 3 **The Checkout Workflow**

from within the bus component. In the Handle method, the saga performs a calculation or data access. It then posts another command to other listening sagas or just fires an event as a notification. For example, imagine the checkout workflow is as shown in **Figure 3**.

The saga performs all steps up to accepting payment. At that point, it pushes an AcceptPayment command to the bus for the PaymentSaga to proceed. The PaymentSaga will run and fire a PaymentCompleted or PaymentDenied event. These events will again be handled by the CheckoutSaga. That saga will then advance to the delivery step with another command placed against another saga interacting with the external subsystem of the shipping partner company.

The concatenation of commands and events keeps the saga live until it reaches completion. In that regard, you could think of a saga as a classic workflow with starting and ending points. Another thing to note is that a saga is usually persistent. Persistence is typically handled by the bus. The sample Bus class presented here doesn't support persistence. A commercial bus such as NServiceBus or even an open source bus like Rebus might use SQL Server. For persistence to occur, you must give a unique ID to each saga instance.

Wrapping Up

For modern applications to be truly effective, they must be able to scale with business requirements. A message-based architecture makes it incredibly easy to extend and modify business workflows and support new scenarios. You can manage extensions in total isolation, All it takes is adding a new saga or a new handler, registering it with the bus at application startup and letting it know how to handle only the messages it needs to handle. The new component will automatically be invoked only when it's time and will work side by side with the rest of the system. It's easy, simple and effective. ■

Dino Esposito *is the co-author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Programming ASP.NET MVC 5" (Microsoft Press, 2014). A technical evangelist for the Microsoft .NET Framework and Android platforms at JetBrains and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents.wordpress.com and on Twitter at twitter.com/despos.*

Thanks to the following technical expert for reviewing this article: *Jon Arne Saeteras*

dtSearch®

Instantly Search Terabytes of Text

dtSearch's document filters support popular file types, emails with multilevel attachments, databases, web data

Highlights hits in all data types; 25+ search options

With APIs for .NET, Java, C++, SQL, etc. SDKs for multiple platforms. See MS Azure tutorials at www.dtsearch.com/azure

Visit dtSearch.com for
- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval® since 1991

dtSearch.com 1-800-IT-FINDS

Open, Create, Convert, Print
& Save Files

from within your *own* applications.

ASPOSE.TOTAL

allows you to process these file formats:

- Word documents
- Excel spreadsheets
- PowerPoint presentations
- PDF documents
- Project documents
- Visio documents
- Outlook emails
- OneNote documents

PDF PPT XLS DOC EML

DOC XLS PPT PDF EML
PNG XML RTF HTML VSD
BMP & barcode images.

SCAN FOR
20% SAVINGS!

ASPOSE
Your File Format APIs

Contact Us:
US: +1 888 277 6734
EU: +44 141 416 1112
AU: +61 2 8003 5926
sales@aspose.com

Helped over 11,000 companies and over 250,000 users work with documents in their applications.

.NET, Java & Cloud

Your File Format APIs

GET STARTED NOW
- Free Trial
- 30 Day Temp License
- Free Support
- Community Forums
- Live Chat
- Blogs
- Examples
- Video Demos

Windows Runtime Components

Over the next few months I'm going to explore the essentials of the Windows Runtime. The purpose is to break down the higher-level abstractions that developers use in the various language projections and toolchains in order to examine how the Windows Runtime works at the application binary interface (ABI)—the boundary between applications and the binary components they rely on to access OS services.

In some ways the Windows Runtime is just the evolution of COM, which was effectively a binary standard for code reuse and continues to be a popular way to build complex applications and OS components. Unlike COM, however, the Windows Runtime is more narrowly focused and is primarily used as the foundation for the Windows API. Application developers will be more inclined to use the Windows Runtime as a consumer of OS components and less likely to write components themselves. Nevertheless, a good understanding of how all of the different classy abstractions are implemented and projected into various programming languages can only help you write more efficient applications and better diagnose interop and performance problems.

One of the reasons why so few developers understand how the Windows Runtime works (other than its rather sparse documentation) is because the tooling and language projections really obscure the underlying platform. This may be natural for the C# developer, but it certainly doesn't make life comfortable for the C++ developer who really wants to know what's going on under the covers. So let's begin by writing a simple Windows Runtime component with Standard C++ using the Visual Studio 2015 developer command prompt.

I'll start with a simple and traditional DLL that exports a couple of functions. If you want to follow along, create a Sample folder and inside of that create a few source files beginning with Sample.cpp:

```
C:\Sample>notepad Sample.cpp
```

The first thing I'll do is take care of unloading the DLL, which I'll call the component from here on. The component should support unload queries via an exported function call, DllCanUnloadNow, and it's the application that controls the unloading with the CoFreeUnusedLibraries function. I won't spend much time on this because this is the same way components were unloaded in classic COM. Because the component isn't statically linked into the application—with a LIB file, for example—but is instead loaded dynamically via the LoadLibrary function, there needs to be some way for the component to be unloaded eventually. Only the component really knows how many outstanding references are being held so the COM runtime can call its DllCanUnloadNow function to determine whether it's safe to unload. Applications can also perform this housekeeping themselves using the CoFreeUnusedLibraries or CoFreeUnusedLibrariesEx functions. The implementation in the component is straightforward. I need a lock that will keep track of how many objects are alive:

```
static long s_lock;
```

Each object can then simply increment this lock in its constructor and decrement it in its destructor. To keep that simple, I'll write a little ComponentLock class:

```
struct ComponentLock
{
  ComponentLock() noexcept
  {
    InterlockedIncrement(&s_lock);
  }

  ~ComponentLock() noexcept
  {
    InterlockedDecrement(&s_lock);
  }
};
```

Any and all objects that should prevent the component from unloading can then simply embed a ComponentLock as a member variable. The DllCanUnloadNow function can now be implemented quite simply:

```
HRESULT __stdcall DllCanUnloadNow()
{
  return s_lock ? S_FALSE : S_OK;
}
```

There are really two types of objects you can create within a component—activation factories, which were called class factories in classic COM, and the actual instances of some particular class. I'm going to implement a simple "Hen" class and I'll start by defining an IHen interface so the hen can cluck:

```
struct __declspec(uuid("28a414b9-7553-433f-aae6-a072afe5cebd")) __declspec(novtable)
IHen : IInspectable
{
  virtual HRESULT __stdcall Cluck() = 0;
};
```

This is a regular COM interface that just happens to derive from IInspectable rather than directly from IUnknown. I can then use the Implements class template I described in the December 2014 issue (msdn.com/magazine/dn879357) to implement this interface, and provide the actual implementation of the Hen class within the component:

```
struct Hen : Implements<IHen>
{
  ComponentLock m_lock;

  virtual HRESULT __stdcall Cluck() noexcept override
  {
    return S_OK;
  }
};
```

An activation factory is just a C++ class that implements the IActivationFactory interface. This IActivationFactory interface provides the single ActivateInstance method, which is analogous to the classic COM IClassFactory interface and its CreateInstance method. The classic COM interface is actually slightly superior in that it allows the caller to request a specific interface directly, whereas the Windows Runtime IActivationFactory simply returns an IInspectable interface pointer. The application is then responsible for calling the IUnknown QueryInterface method to retrieve a more useful interface to the object. Anyway, it makes the ActivateInstance method quite simple to implement:

```
struct HenFactory : Implements<IActivationFactory>
{
  ComponentLock m_lock;

  virtual HRESULT __stdcall ActivateInstance(IInspectable ** instance)
    noexcept override
  {
    *instance = new (std::nothrow) Hen;
    return *instance ? S_OK : E_OUTOFMEMORY;
  }
};
```

The component allows applications to retrieve a specific activation factory by exporting another function called DllGetActivationFactory. This, again, is analogous to the DllGetClassObject exported function that supports the COM activation model. The main difference is that the desired class is specified with a string rather than a GUID:

```
HRESULT __stdcall DllGetActivationFactory(HSTRING classId,
  IActivationFactory ** factory) noexcept
{
}
```

C# compilers don't know how to parse C++ header files, so I need to provide some metadata with which the C# compiler will be happy.

An HSTRING is a handle that represents an immutable string value. This is the class identifier, perhaps "Sample.Hen," and indicates which activation factory should be returned. At this point there are a number of reasons why calls to DllGetActivationFactory might fail, so I'll start by clearing the factory variable with a nullptr:

```
*factory = nullptr;
```

Now I need to get the backing buffer for the HSTRING class identifier:

```
wchar_t const * const expected = WindowsGetStringRawBuffer(classId, nullptr);
```

I can then compare this value with all of the classes my component happens to implement. So far there's only one:

```
if (0 == wcscmp(expected, L"Sample.Hen"))
{
  *factory = new (std::nothrow) HenFactory;
  return *factory ? S_OK : E_OUTOFMEMORY;
}
```

Otherwise, I'll return an HRESULT indicating that the requested class isn't available:

```
return CLASS_E_CLASSNOTAVAILABLE;
```

That's all the C++ I need to get this simple component up and running, but there's still a bit more work to do in order to actually make a DLL for this component and then describe it to those pesky C# compilers that don't know how to parse header files. To make a DLL, I need to involve the linker, specifically its ability to define the functions exported from the DLL. I could use the Microsoft compiler-specific dllexport __declspec specifier, but this is one of the rare cases where I prefer to talk to the linker directly and instead provide a module-definition file with the list of exports. I find this approach less error prone. So it's back to the console for the second source file:

```
C:\Sample>notepad Sample.def
```

This DEF file simply needs a section called EXPORTS that lists the functions to be exported:

```
EXPORTS
DllCanUnloadNow         PRIVATE
DllGetActivationFactory PRIVATE
```

I can now provide the C++ source file along with this module-definition file to the compiler and linker to produce the DLL, and then use a simple batch file as a convenience to build the component and place all of the build artifacts in a subfolder:

```
C:\Sample>type Build.bat
@md Build 2>nul

cl Sample.cpp /nologo /W4 /FoBuild\ /FeBuild\Sample.dll /link /dll /def:Sample.def
```

I'll gloss over the deep magic that is the batch file scripting language and focus on the Visual C++ compiler options. The /nologo option suppresses the display of the copyright banner. The option is also forwarded to the linker. The indispensable /W4 option tells the compiler to display more warnings for common coding bugs. There's no /FoBuild option. The compiler has this hard-to-read convention whereby output paths follow the option, in this case /Fo, without a space in between. Anyway, the /Fo option is used to coerce the compiler to dump the object file in the Build subfolder. It's the only build output that doesn't default to the same output folder as the executable defined with the /Fe option. The /link option tells the compiler that subsequent arguments are to be interpreted by the linker. This avoids having to call the linker as a secondary step and, unlike the compiler, the linker's options are case-insensitive and do employ a separator between the name of an option and any value, as is the case with the /def option that indicates the module-definition file to use.

I can now build my component quite simply and the resulting Build subfolder contains a number of files, only one of which matters. Naturally, that's the Sample.dll executable that can be loaded into the application's address space. But this is not enough. An application developer needs some way to know what the component contains. A C++ developer would probably be satisfied with a header file including the IHen interface, but even that isn't particularly convenient. The Windows Runtime includes the concept of language projections whereby a component is described in such a way that different languages can discover and project its types into their programming models. I'll explore language projection in the coming months, but for now let's just get this sample to work from a C# application because that's the most convincing. As I mentioned, C# compilers don't know how to parse C++ header files, so I need to provide some metadata with which the C# compiler will be happy. I need to produce a WINMD file that contains the CLR metadata describing my component. This is no simple matter because the native types I might use for the component's ABI can often look very different when projected into C#. Fortunately, the Microsoft IDL compiler has been repurposed to produce a WINMD file, given an IDL file that uses a few new keywords. So it's back to the console for our third source file:

```
C:\Sample>notepad Sample.idl
```

First, I need to import the definition of the prerequisite IInspectable interface:

```
import "inspectable.idl";
```

I can then define a namespace for the component's types. This must match the name of the component itself:

```
namespace Sample
{
}
```

Now I need to define the IHen interface I previously defined in C++, but this time as an IDL interface:

```
[version(1)]
[uuid(28a414b9-7553-433f-aae6-a072afe5cebd)]
interface IHen : IInspectable
{
  HRESULT Cluck();
}
```

This is good old IDL and if you've used IDL in the past to define COM components, you shouldn't be surprised by any of this. All Windows Runtime types must, however, define a version attribute. This used to be optional. All interfaces must also derive from IInspectable directly. There's effectively no interface inheritance in the Windows Runtime. This has some negative consequences that I'll talk about in the coming months.

The default interface is the interface that takes the place of parameters and return types when those types specify the class itself.

And, finally, I need to define the Hen class itself using the new runtimeclass keyword:

```
[version(1)]
[activatable(1)]
runtimeclass Hen
{
  [default] interface IHen;
}
```

Again, the version attribute is required. The activatable attribute, while not required, indicates that this class may be activated. In this case, it indicates that default activation is supported via the IActivationFactory ActivateInstance method. A language projection should present that as a C++ or C# default constructor or whatever makes sense to a particular language. Finally, the default attribute before the interface keyword indicates that the IHen interface is the default interface for the Hen class. The default interface is the interface that takes the place of parameters and return types when those types specify the class itself. Because the ABI only trades in COM interfaces and the Hen class is not itself an interface, the default interface is its representative at the ABI level.

There's a lot more to explore here but this will do for the moment. I can now update my batch file to produce a WINMD file describing my component:

```
@md Build 2>nul

cl Sample.cpp /nologo /W4 /FoBuild\ /FeBuild\Sample.dll /link /dll /def:Sample.def

"C:\Program Files (x86)\Windows Kits\10\bin\x86\midl.exe" /nologo /winrt
/out %~dp0Build /metadata_dir "c:\Program Files (x86)\Windows Kits\10\
References\Windows.Foundation.FoundationContract\1.0.0.0" Sample.idl
```

I'll again gloss over the magic in the batch file and focus on what's new with the MIDL compiler options. The /winrt option is the key and indicates that the IDL file contains Windows Runtime types rather than traditional COM or RPC-style interface definitions. The /out option just ensures that the WINMD file resides in the same folder as the DLL as this is required by the C# toolchain. The /metadata_dir option tells the compiler where it can find the metadata that was used to build the OS. As I write this, the Windows SDK for Windows 10 is still settling down and I need to be careful to invoke the MIDL compiler that ships with the Windows SDK and not the one provided by the path in the Visual Studio tools command prompt.

Running the batch file now produces both the Sample.dll and the Sample.winmd, which I can then reference from a C# Windows Universal app and use the Hen class as if it were just another CLR library project:

```
Sample.Hen h = new Sample.Hen();
h.Cluck();
```

The Windows Runtime is built on the foundations of COM and Standard C++. Concessions have been made to support the CLR and make it very easy for C# developers to use the new Windows API without requiring any interop components. The Windows Runtime is the future of the Windows API.

I specifically presented the development of a Windows Runtime component from the perspective of classic COM and its roots in the C++ compiler so that you can understand from where this technology comes. However, this approach quickly becomes rather impractical. The MIDL compiler actually provides far more than just the WINMD file and we can actually use it, among other things, to generate the canonical version of the IHen interface in C++. I hope you'll join me next month as we explore a more reliable workflow for authoring Windows Runtime components and also solve a few interop problems along the way. ■

Kenny Kerr is *a computer programmer based in Canada, as well as an author for Pluralsight and a Microsoft MVP. He blogs at kennykerr.ca and you can follow him on Twitter at twitter.com/kennykerr.*

Thanks to the following Microsoft technical expert for reviewing this article: *Larry Osterman*

DOMAINS | MAIL | HOSTING | eCOMMERCE | SERVERS

1&1 DEDICATED SERVER BUSINESS LINE

Trust is important when it comes to choosing the right server provider. With 13 years of server experience and 6,000 employees in 11 countries, 1&1 is one of the largest Internet service providers in the world and a company you can rely on. Benefit from our expertise and the maximum security offered by our high-tech data centers.

Dedicated Server Business Line X8i and X10i, built on Dell™ PowerEdge™ R630 hardware

- Latest Intel® Xeon® processors E5-2600 V3 (from 8 cores HT/2.3 GHz) and from 64 GB DDR4 RAM
- From 4 TB HDD, Hardware RAID 6
- 1 Gbit/s connection with unlimited traffic
- Maximum security due to redundant components
- Optional 240 GB Intel® SSD hard drive with RAID 1
- Broad range of Linux and Windows operating systems
- High-tech data centers
- Best in Class Cisco-based IP firewall
- Maximum availability
- 24/7 Customer Support
- Also available as a 1&1 Managed Server with OS managed by our experts
- LIMITED TIME OFFER: Additional SSD for free*

The complete 1&1 server range: Great entry-level web servers from $39.99 per month, to high-end servers with the highest capabilities.

Windows

CHECK OUT ALL OUR DEDICATED SERVER OFFERS AT **1and1.com**

1 TRIAL TRY FOR 30 DAYS

1 MONTH FLEXIBLE PAYMENT OPTIONS

1 CALL SPEAK WITH AN EXPERT 24/7

1 (877) 461-2631

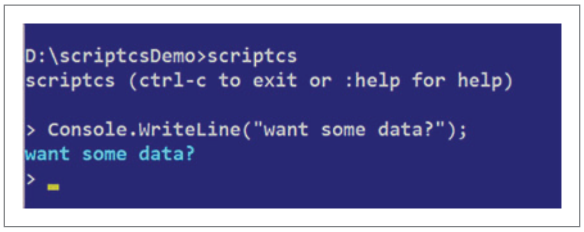
1&1®

1and1.com

*SSD offer and promotional pricing applies to the initial minimum contract term only and will renew at the regular rates. Visit www.1and1.com for full offer details, terms and conditions. Dell, the Dell logo, the Dell badge and PowerEdge are trademarks of Dell Inc. Intel, the Intel Logo, Intel Inside, the Intel Inside logo, Intel. Experience What's Inside are trademarks of Intel Corporation in the U.S. and/or other countries. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are property of their respective owners. ©2015 1&1 Internet. All rights reserved.

Exploring Entity Framework Behavior at the Command Line with Scriptcs

I spend a lot of time trying to educate people about how Entity Framework behaves with regard to relationships and disconnected data. EF doesn't always follow the rules you might imagine in your head, so I always have the goal to help you be aware of what to expect, why some things work in a certain way and how to work with or avoid a particular behavior.



```
D:\scriptcsDemo>scriptcs
scriptcs (ctrl-c to exit or :help for help)

> Console.WriteLine("want some data?");
want some data?
>
```

Figure 1 **Running C# Code at the Command Line in the Scriptcs REPL Environment**

When I want to demonstrate something to developers, I have a handful of workflow options. For example, I can build integration tests and run them one at a time as I discuss each behavior. What I don't like about this path is I often want to debug so I can drill into objects and show what's going on under the covers. But debugging automated tests can be slow because all of the source Visual Studio needs to load.

Another workflow I've used involves building a console app and debugging through it, stopping at break points and inspecting objects. This is also not as interactive as I'd like because all of the code needs to be there in advance.

What I really want to be able to do is type some code, execute it so my audience can see its effect, then modify that code and execute it again to highlight the different response. For example, I've used this method as in the following code, to show the difference between this:

```
using (var context = new AddressContext()) {
  aRegionFromDatabase=context.Regions.FirstOrDefault();
  context.Address.Add(newAddress);
  newAddress.Region=aRegionObjectFromDatabase;
}
```

this:

```
using (var context = new AddressContext()) {
  context.Address.Add(newAddress);
  newAddress.Region=aRegionObjectFromDatabase;
}
```

and this:

```
newAddress.Region=aRegionObjectFromDatabase;
using (var context = new AddressContext()) {
  context.Address.Add(newAddress);
}
```

The difference, by the way, is that in the first example, where EF retrieves the Region in the same context instance, EF will understand that the region pre-exists and will not try to re-add it to the database. In the second and third cases, EF rules will infer that the Region, like its "parent" Address, is new and will add it into the database. In the long run, I recommend using a foreign key value and not an instance. It's really helpful to be able to demonstrate cause and effect.

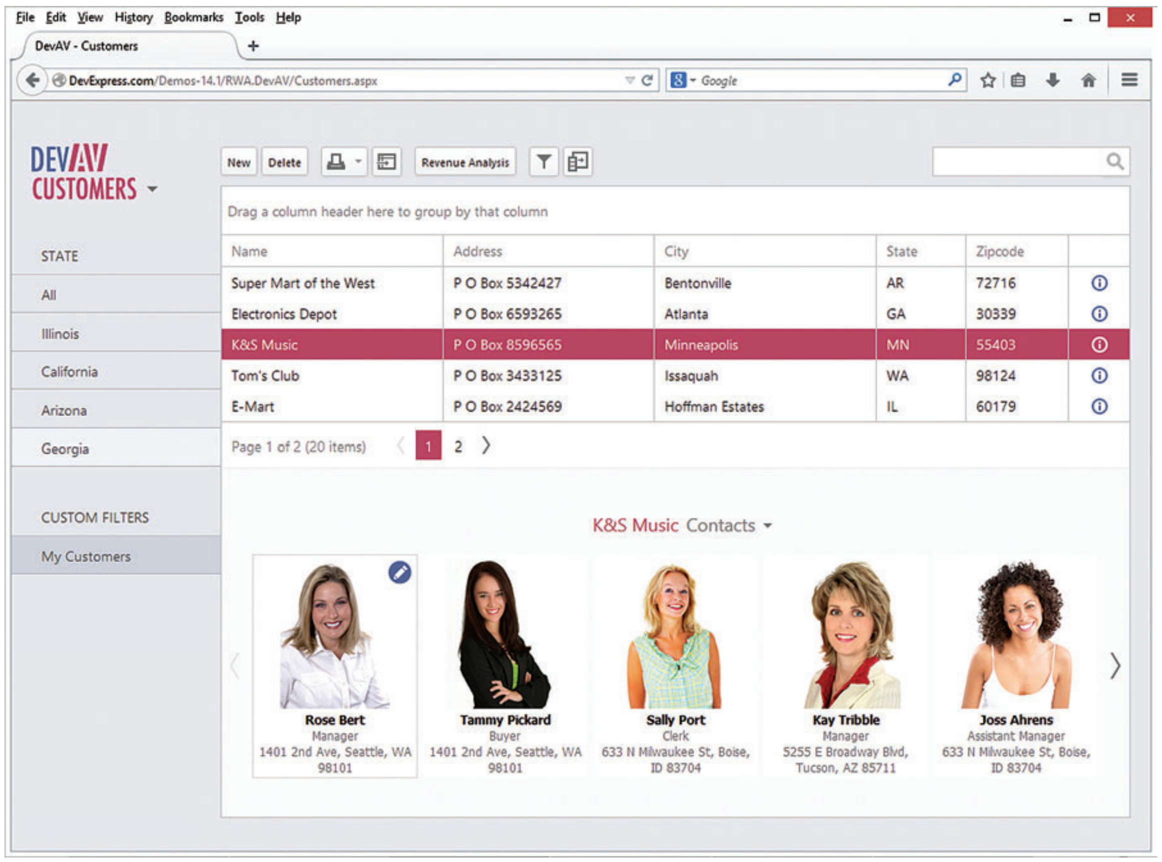
Code download available at msdn.microsoft.com/magazine/msdnmag0715.

Sure, I could demonstrate this effect of with tests, but it's a little convoluted. And there's the argument that tests are not to prove theories, but to validate code. I also don't like showing differences with a console app, because you have to run the app after each change to the of code. A better alternative is to use the fantastic LinqPad application, and I've done that a few times. But now I'm leaning toward a fourth way to get to my happy demo place, with Scriptcs.

Scriptcs also provides a very lightweight mechanism for creating scripts with C# in whatever text editor you want and then running those scripts from the command line.

Scriptcs (scriptcs.net) has been around since 2013 and is completely open source. It was written on top of Roslyn to provide a command-line runtime that lets you run C# code outside of Visual Studio. Scriptcs also provides a very lightweight mechanism for creating scripts with C# in whatever text editor you want and then running those scripts from the command line.

I'd often heard of Scriptcs because one of the key people behind it is Glenn Block, for whom I have enormous respect. But I never looked too closely at it until very recently, after listening to Block talking about the bright future of Scriptcs on an episode of DotNetRocks (bit.ly/1AA1m4z). My first thought on seeing this tool was that it could allow me to perform interactive demos right at the



Name	Address	City	State	Zipcode
Super Mart of the West	P O Box 5342427	Bentonville	AR	72716
Electronics Depot	P O Box 6593265	Atlanta	GA	30339
K&S Music	P O Box 8596565	Minneapolis	MN	55403
Tom's Club	P O Box 3433125	Issaquah	WA	98124
E-Mart	P O Box 2424569	Hoffman Estates	IL	60179

Page 1 of 2 (20 items) 1 2

Rose Bert	Tammy Pickard	Sally Port	Kay Tribble	Joss Ahrens
Manager	Buyer	Clerk	Manager	Assistant Manager
1401 2nd Ave, Seattle, WA 98101	1401 2nd Ave, Seattle, WA 98101	633 N Milwaukee St, Boise, ID 83704	5255 E Broadway Blvd, Tucson, AZ 85711	633 N Milwaukee St, Boise, ID 83704

Your Next Great Web App Starts Here

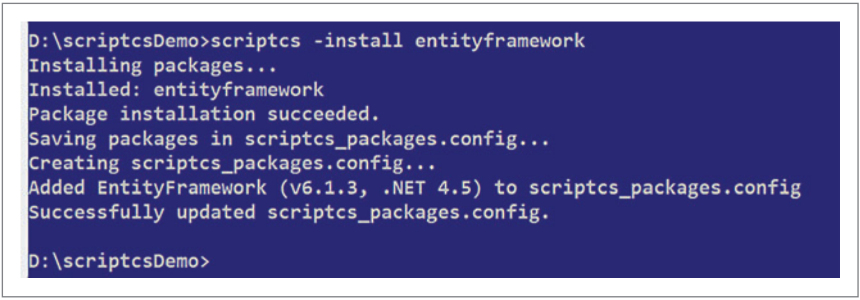
With DevExpress web controls, you can build a bridge to the future on the platform you know and love. The 95+ AJAX Web Forms Controls & 50+ MVC Extensions that ship inside the DevExpress ASP.NET Subscription allow you to create functional, elegant and interactive experiences for the web, regardless of the target browser or computing device. All major browsers including Internet Explorer, FireFox, Chrome, Safari and Opera, are fully supported and continuously tested to ensure the highest compatibility.

Get started today.

Download your free 30-day trial and experience the DevExpress difference yourself.
devexpress.com/asp

DevExpress®

All trademarks or registered trademarks are property of their respective owners.



```
D:\scriptcsDemo>scriptcs -install entityframework
Installing packages...
Installed: entityframework
Package installation succeeded.
Saving packages in scriptcs_packages.config...
Creating scriptcs_packages.config...
Added EntityFramework (v6.1.3, .NET 4.5) to scriptcs_packages.config
Successfully updated scriptcs_packages.config.

D:\scriptcsDemo>
```

Figure 2 **Installing NuGet Packages into Scriptcs**

command line. And by combining it with EF, I can rationalize sharing Scriptcs with readers of this *data*-focused column.

A Tiny Intro to Scriptcs

There are lots of great resources for Scriptcs. I am far from an expert, so I'll just give some highlights and point you to the scriptcs.net site for more info. I also found the short video from Latish Sengal at bit.ly/1R6mF8s to be a perfect first look at Scriptcs.

First, you'll need to install Scriptcs onto your development machine using Chocolatey, which means you'll also need Chocolatey installed on your machine. If you don't already have Chocolatey, it's an incredible tool for installing software tools. Chocolatey uses NuGet to install tools (and the tools or APIs they depend on) in the same way that NuGet uses packages to deploy assemblies.

Scriptcs provides a Read, Evaluate, Play, Loop (REPL) environment, which you can think of as akin to a runtime environment. You can use Scriptcs to execute C# commands one by one at the command line, or to execute Scriptcs script files created in a text editor. There are even Scriptcs IntelliSense extensions for some editors. The best way to see Scriptcs in action is to start with line-by-line execution. Let's start with that.

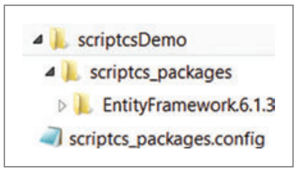
Once Scriptcs is installed, navigate to the folder where you want any saved code to be stored, then execute the scriptcs command. This will start the REPL environment and inform you that Scriptcs is running. It also returns a > prompt.

At the prompt, you can type any C# command that's found in a few of the most common .NET namespaces. These are all available right off the bat and are from the same assemblies that a typical .NET console application project has by default. **Figure 1** shows Scriptcs starting at the command prompt and then executing a line of C# code, as well as the results that are displayed.

Scriptcs has a handful of directives that let you do things like reference an assembly (#r) or load an existing script file (#load).

Another cool feature is that Scriptcs lets you easily install NuGet packages. You do this at the command prompt using the -install parameter of the Scriptcs command. For example, **Figure 2** shows what happens when I install Entity Framework.

There's more to package installs, though, as you can see from the screenshot of my folder after I've installed Entity Framework (see **Figure 3**)—Scriptcs has created a scriptcs_packages folder along with the contents of the relevant package.



```
◢ scriptcsDemo
  ◢ scriptcs_packages
    ▷ EntityFramework.6.1.3
  scriptcs_packages.config
```

Figure 3 **The Scriptcs NuGet Package Installer Creates Familiar Package Folders and Config Files**

Creating a Model and Some Common Setup Code

I'll be doing my experiments against a particular model I already built in Visual Studio using code first. I have one assembly with my classes—Monkey, Banana and Country—and another assembly that contains a MonkeysContext that inherits from the EF DbContext and exposes DbSets of Monkeys, Bananas and Countries. I verified that my model was set up correctly using the View Entity Data Model feature of the Entity Framework Power Tools extension for Visual Studio (bit.ly/1K8qhkO). This way, I know I can depend on the compiled assemblies I'll be using in my command-line tests.

In order to do these experiments, there's some common setup I have to perform. I need references to my custom assemblies, as well as to the EF assembly, and I need code that instantiates new monkey, country and MonkeysContext objects.

Scriptcs provides a Read, Evaluate, Play, Loop (REPL) scenario, which you can think of as akin to a runtime environment.

Rather than repeat this setup continually at the command prompt, I'll create a script file for Scriptcs. Script files, which have a .csx extention, are the more common way to take advantage of Scriptcs. I first used Notepad++ with the Scriptcs plug-in, but was encouraged to try out Sublime Text 3 (sublimetext.com/3). Doing so allowed me to use not only the Scriptcs plug-in, but also OmniSharp, a C# IDE plug-in for Sublime Text 3, which provides an amazing coding experience considering you're in a text editor that supports OSX and Linux, as well as Windows.

OmniSharp lets you build Scriptcs, as well as build for many other systems. With Sublime set up, I'm ready to create a .csx file that encapsulates the common setup code I want for testing out my model.

In the text editor, I create a new file, SetupForMonkeyTests.csx, add some Scriptcs commands for referencing needed assemblies, and then add the C# code to create some objects, as shown here:

```
#r "..\DataModel Assemblies\DataModel.dll"
#r "..\DataModel Assemblies\DomainClasses.dll"
#r "..\scriptcs_packages\EntityFramework.6.1.3\lib\net45\
EntityFramework.dll"

using DomainClasses;
using DataModel;
using System.Data.Entity;

var country=new Country{Id=1,Name="Indonesia"};
var monkey=Monkey.Create("scripting gal", 1);
Database.SetInitializer(new NullDatabaseInitializer<MonkeysContext>());
var ctx=new MonkeysContext();
```

```
40, 0); AddEAN13Sup5(page.Elements, 480, 0); AddUPCVersionA(page.Elements, 0, 135); AddUPCVersionASup2(page.Elements, 240, 135); AddUPCVersionASup5(page.Elements, 480, 135); AddEAN8(page.Elemen
ements, 480, 270); AddUPCVersionE(page.Elements, 0, 405); AddUPCVersionESup2(page.Elements, 240, 405); AddUPCVersionESup5(page.Elements, 480, 405); // Add the page to the document document.Pages.Add
tionAndRectangle(elements, "EAN/JAN 13 Bar Code", x, y, 204, 99); BarCode barCode = new Ean13("123456789012", x, y + 21); barCode.X += (204 - barCode.GetSymbolWidth()) / 2; elements.Add(barCode); } privat
ctangle(elements, "EAN/JAN 13 Bar Code, 2 digit supplement", x, y, 204, 99); BarCode barCode = new Ean13Sup2("12345678901212", x, y + 21); barCode.X += (204 - barCode.GetSymbolWidth()) / 2; elements.Add(
float x, float y) { AddCaptionAndRectangle(elements, "EAN/JAN 13 Bar Code, 5 digit supplement", x, y, 204, 99); BarCode barCode = new Ean13Sup5("12345678901212345", x, y + 21); barCode.X += (204 - barCode.
PCVersionA(Group elements, float x, float y) { AddCaptionAndRectangle(elements, "UPC Version A Bar Code", x, y, 204, 99); BarCode barCode = new UpcVersionA("12345678901", x, y + 21); barCode.X += (204 - ba
PCVersionASup2(Group elements, float x, float y) { AddCaptionAndRectangle(elements, "UPC Version E Bar Code, 2 digit supplement", x, y, 204, 99); BarCode barCode = new UpcVersionASup2("1234567890112", x,
dd(barCode); } private void AddUPCVersionASup5(Group elements, float x, float y) { AddCaptionAndRectangle(elements, "UPC Version E Bar Code, 5 digit supplement", x, y, 204, 99); BarCode barCode = new UpcVe
GetSymbolWidth()) / 2; elements.Add(barCode); } private void AddEAN8(Group elements, float x, float y) { AddCaptionAndRectangle(elements, "EAN/JAN 8 Bar Code", x, y, 204, 99); BarCode barCode = new Ean8("12
fileDialog.Title = "Open File Dialog";  fileDialog.Filter = "Adobe PDF files (*.pdf)|*.pdf|All Files (*.*)|*.*";  if (fileDialog.ShowDialog() == DialogResult.OK)  {  pdfViewer.OpenFile(fileDialog.FileName, "");  }  SaveFileDialog sa
e Dialog";  saveFileDialog.Filter = "Adobe PDF files (*.pdf)|*.pdf|All Files (*.*)|*.*";  if (saveFileDialog.ShowDialog()==DialogResult public ViewerForm().OK)  {  pdfViewer.SaveAs(saveFileDialog.FileName);  }  if (pdfViewer.P
Dialog();  }  else  {  MessageBox.Show("Please open a file to print");  }   OpenFileDialog fileDialog = new OpenFileDialog();   file{ Dialog.Title = "Open File Dialog";  fileDialog.InitialDirectory = @"c:\";  fileDialog.Filter = "
DialogResult.OK)  {  DynamicPDFViewerClass test = new DynamicPDFViewerClass();  PDFPrinter printer = test.OpenFileForPrinter// Convert RTF file to PDF (fileDialog.FileName);  printer.PrintQuiet();  }  byte[] content
rvices;  GCHandle gch = GCHandle.Alloc(contents, GCHandleType.Pinned);   IntPtr contentsIntPtr =gch.AddrOfPinnedObject();pdf Converter.Convert(@"C:\Input.rtf", @"C:\Ouptut.pdf"); Viewer.OpenBuffer(contentsInt
ark Page Element:", x, y); pageElements.Add(new Bookmark("Bookmarked Text", x + 5, y + 20, parentOutline)); pageElements.Add  // Output PDF to Custom Viewer (new Label("This text is bookmarked.", x + 5, y + 2
Elements, float x, float y) { // Adds a circlt to the pageElements AddCaptionAndRectangle(pageElements, "Circle Page Element:", x, pdfViewer.OpenFile(@"C:\Ouptut.pdf", ""); y); pageElements.Add(new Circle(x + 112
arge)); } private void AddFormattedTextArea(Group pageElements, float x, float y) { // Adds a formatted text area to thepageEle}ments string formattedHtml = "<p><i>Dynamic</i><b>PDF</b>&tm; Generator v6.0 for
tting support for text that appears in the document. You have " + "complete control over 8 paragraph properties: spacing before, spacing after, first line " + "indentation, left indentation, right indentation, alignment, all
nt face='Times'>font face, </font><font pointSize='6'>font " + "size, </font><font color='FF0000'>color, </font><b>bold, </b><i>italic and </i><u>underline</u>; " + "and 2 line properties: leading, and leading type. 
rea = new FormattedTextArea(formattedHtml, x + 5, y + 20, 215, 60, FontFamily.Helvetica, 9, false); // Sets the indent property formattedTextArea.Style.Paragraph.Indent = 18; AddCaptionAndRectangle(pageElement
Elements, "FormattedTextArea Overflow Text:", x + 279, y); pageElements.Add(formattedTextArea); // Create an overflow formatted text area for the overflow text FormattedTextArea overflowFormattedTextArea = form
284, y + 20); pageElements.Add(overflowFormattedTextArea); } private void AddImage(Group pageElements, float x, float y) { // Adds an image to the pageElements AddCaptionAndRectangle(pageElements, "Image
DPDFLogo.png"), x + 112.5f, y + 50f, 0.24f); // Image is sized and centered in the rectangle image.SetBounds(215, 60); image.VAlign = VAlign.Center; image.Align = Align.Center; pageElements.Add(image); } private vo
eElements AddCaptionAndRectangle(pageElements, "Label & PageNumberingLabel Page Elements:", x, y); string labelText = "Labels can be rotated"; string numberText = "PageNumberingLabels contain page numbe
x + 5, y + 12, 220, 80, Font.TimesRoman, 12, TextAlign.Center); label.Angle = 8; PageNumberingLabel pageNumLabel = new PageNumberingLabel(numberText, x + 5, y + 55, 220, 80, Font.TimesRoman, 12, TextAlign
ts.Add(label); } private void AddLine(Group pageElements, float x, float y) { // Adds a line to the pageElements AddCaptionAndRectangle(pageElements, "Line Page Element:", x, y); pageElements.Add(new Line(x + 5,
ne(x + 220, y + 20, x + 5, y + 80, 3, RgbColor.Green)); } private void AddLink(Group pageElements, float x, float y) { // Adds a link to the pageElements Font font = Font.TimesRoman; string text = "This is a link to Dyna
t:", x, y); Label label = new Label(text, x + 5, y + 20, 215, 80, font, 12, RgbColor.Blue); label.Underline = true; Link link = new Link(x + 5, y + 20, font.GetTextWidth(text, 12), 12 - font.GetDescender(12), new UrlAction("h
ments.Add(link); } private void AddPath(Group pageElements, float x, float y) { // Adds a path to the pageElements ceTe.DynamicPDF.PageElements.Path path = new ceTe.DynamicPDF.PageElements.Path(x + 5, y + 2
hs.Add(new LineSubPath(x + 215, y + 40)); path.SubPaths.Add(new CurveToSubPath(x + 108, y + 80, x + 160, y + 80)); path.SubPaths.Add(new CurveSubPath(x + 5, y + 40, x + 65, y + 80, x + 5, y + 60)); AddCaption
(path); } private void AddRectangle(Group pageElements, float x, float y) orderedList = orderedList.GetOverFlowList(x + 5, y + 20); AddCaptionAndRectangle(page.Elements, "Ordered List Page Element Overflow:", x,
// Create an unordered list UnorderedList unorderedList = new UnorderedList(x + 5, y + 20, 400, 90, Font.Helvetica, 10); unorderedList.Items.Add("Fruits"); unorderedList.Items.Add("Vegetables"); UnorderedSubList un
unorderedSubList.Items.Add(" Citrus"); unorderedSubList.Items.Add(" Non-Citrus"); AddCaptionAndRectangle(page.Elements, "Unordered List Page Element:", x, y, 225, 110); UnorderedSubList unorderedSubList2 =
SubList2.Iten                                                                                              s[0].SubLists.AddUnorderedSubList(); subUnorderedSubList.Items.Add("Lime
subUnordere                                                                                                norderedSubList2.Items.Add("Banana"); UnorderedSubList subUnorderedSub
subUnorderec                                                                                               UnorderedSubList(); subUnorderedSubList4.Items.Add("String Bean"); subUn
"Kidney Bean                                                                                               AndRectangle(page.Elements, "Unordered List Page Element Overflow:", x, y,
AddTextField(                                                                                              his is a Scrollable TextField"; txt.BorderColor = RgbColor.Black; txt.Backgroun
t); TextField tx                                                                                           gth = 9; txt1.BorderColor = RgbColor.Black; txt1.BackgroundColor = RgbColo
es(); pieSeries.DataLabel = da; plotArea.Series.Add(pieSeries); pieSeries.Elements.Add(27, "Website A"); pieSeries.Elements.Add(19, "Website B"); pieSeries.Elements.Add(21, "Website C"); pieSeries.Elements[0].Colo
lements[2].Co                                                                                  ectangle(pageElements, "TextField Form Page Element:", x, y, 504, 85); } private void AddCo
nboBox("cmb                                                                                    cb.Font = Font.Helvetica; cb.FontSize = 12; cb.Items.Add("Item 1"); cb.Items.Add("Item 2")
ble"); cb.Item                                                                                 ComboBox cb1 = new ComboBox("cmb1name", x + 303, y + 40, 150, 20); cb1.BorderColor
nt.Helvetica;                                                                                  "Item 4"); cb1.Items.Add("Non-Editable"); cb1.Items["Non-Editable"].Selected = true; cb1.E
Add(cb1); Cor                                                                                  put.pdf");System.Diagnostics.Process.Start("Output.pdf"); AsyncConverter aConverter = ne
Converter_Co                                                                                   nverter.Convert(@"C:\temp\DocumentA.rtf", @"C:\temp\OutputA.pdf"); aConverter.Convert(
(@"C:\temp\DocumentC.rtf", @"C:\temp\OutputC.pdf"); aConverter.Convert("http://www.yahoo.com", @"C:\Temp\yahoo.pdf"); ConversionOptions options = new ConversionOptions(720, 720, 72, true); ceTe.Dynamic
p\\output.pdf", options); ceTe.DynamicPDF.Conversion.Converter.Convert("C:\\temp\\Document2.docx", "C:\\temp\\output.pdf", options); string sampleHtml = "<html><body><p>This is a very simple HTML string inc
le border=\"1\"><tr><td>100</td><td>200</td>" + "<td>300</td></tr><tr><td>400</td><td>500</td><td>600</td></tr></table></body></html>";Conversion.Converter.ConvertHtmlString(sampleHtml, "C:\\temp\
ame", Path.Combine(GetPath(), "LetterPortrait.pdf")); printJob.DocumentName = "Letter Portrait"; if (printJob.Printer.Color) printJob.PrintOptions.Color = true; if (printJob.Printer.Collate) printJob.PrintOptions.Collate =
```

DEVELOPED FOR

INTUITIVE USE

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

TRY OUR PDF SOLUTIONS FREE TODAY!

www.DynamicPDF.com/eval or call 800.631.5006 | +1 410.772.8620

ceTesoftware

Figure 4 **Starting Scriptcs; Loading a .csx File; Checking References and Existing Variables**

```
D:\ScriptCS Demo>scriptcs
scriptcs (ctrl-c to exit or :help for help)

> #load "SetupForMonkeyTests.csx"
> :references
[
  "System",
  "System.Core",
  "System.Data",
  "System.Data.DataSetExtensions",
  "System.Xml",
  "System.Xml.Linq",
  "System.Net.Http",
  "C:\\Chocolatey\\lib\\scriptcs.0.14.1\\tools\\ScriptCs.Core.dll",
  "C:\\Chocolatey\\lib\\scriptcs.0.14.1\\tools\\ScriptCs.Contracts.dll",
  "D:\\ScriptCS Demo\\scriptcs_packages\\EntityFramework.6.1.3\\lib\\
    net45\\EntityFramework.dll",
  "D:\\ScriptCS Demo\\scriptcs_packages\\EntityFramework.6.1.3\\lib\\
    net45\\EntityFramework.SqlServer.dll",
  "D:\\ScriptCS Demo\\DataModel Assemblies\\DataModel.dll",
  "D:\\ScriptCS Demo\\DataModel Assemblies\\DomainClasses.dll"
]
> :vars
[
  "DomainClasses.Country country = DomainClasses.Country",
  "DomainClasses.Monkey monkey = DomainClasses.Monkey",
  "DataModel.MonkeysContext ctx = DataModel.MonkeysContext"
]
>
```

Remember, the #r commands are Scriptcs commands to add references to needed assemblies. A number of oft-used System assemblies are referenced by default in order to provide an out-of-the-box experience similar to the one you get when you create a new console project in Visual Studio. So I don't need to specify those, but I do need to reference my custom assemblies, as well as the EntityFramework assembly installed by NuGet. The rest is just C# code. Notice that I'm not defining classes here. This is a script, so Scriptcs will just read and execute line by line whatever is in the file. With the OmniSharp Sublime Text combo, I can even build to ensure my syntax is correct.

With this file in hand, I'll return to the command line and check out some EF behavior using my model and classes.

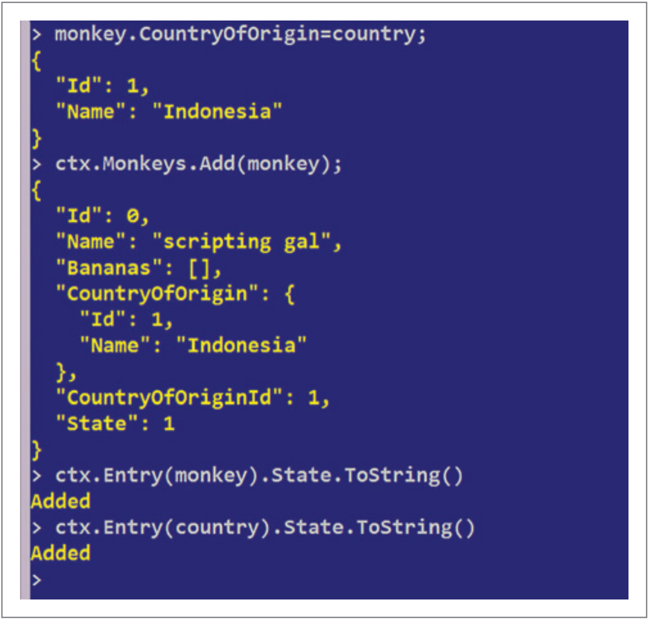
On to My Command-Line Experiments with EF

Back at the command prompt, I'll start the Scriptcs REPL with just the scriptcs command and no parameters.

Once the REPL is active, the first thing I want it to do is load the .csx file. Then I'll use the :references and :vars commands to verify the REPL correctly ran the .csx file. (Scriptcs has a number of REPL commands that start with a colon. You can see a list by typing :help.) **Figure 4** shows my session so far; you can see all of the APIs that are referenced, as well as the objects I created.

I can also inspect the objects by just typing the variable name at the prompt. Here, for example, is my monkey object:

```
> monkey
{
  "Id": 0,
  "Name": "scripting gal",
  "Bananas": [],
  "CountryOfOrigin": null,
  "CountryOfOriginId": 1,
  "State": 1
}
>
```



```
> monkey.CountryOfOrigin=country;
{
  "Id": 1,
  "Name": "Indonesia"
}
> ctx.Monkeys.Add(monkey);
{
  "Id": 0,
  "Name": "scripting gal",
  "Bananas": [],
  "CountryOfOrigin": {
    "Id": 1,
    "Name": "Indonesia"
  },
  "CountryOfOriginId": 1,
  "State": 1
}
> ctx.Entry(monkey).State.ToString()
Added
> ctx.Entry(country).State.ToString()
Added
>
```

Figure 5 **Using Scriptcs to See Immediately How EF Responds to the DbSet.Add Method with a Graph**

Now I'm ready to start exploring some EF behavior. Returning to my earlier examples, I'll check how EF responds to attaching the country object to the monkey when the context is or isn't tracking the monkey and when it is or isn't tracking the country.

EF only changes the state of the related object when its state is unknown.

In the first test, I'll attach the pre-existing country to the monkey's CountryOfOrigin property, then Add the monkey to the context. Finally, I'll use the DbContext.Entry().State property to examine how EF understands the objects' state. It makes sense that the monkey is Added, but notice that EF thinks the country is Added, as well. That's how EF treats a graph. Because I used the Add method on the root of the graph (monkey), EF is marking everything in the graph as Added. When I call SaveChanges, the country will get inserted into the database table and, as you can see in

Figure 6 **Manually Assigning the Unchanged State**

```
> :reset
> #load "SetupForMonkeyTests.csx"
> ctx.Entry(country).State=EntityState.Unchanged;
2
> ctx.Monkeys.Add(monkey);
{...response...}
> monkey.CountryOfOrigin=country;
{...response...}
> ctx.Entry(monkey).State.ToString()
Added
> ctx.Entry(country).State.ToString()
Unchanged
>
```

Figure 5, I'll have two rows for Indonesia. The fact that country already had a key value (Id) will be ignored.

Next, I'll use the :reset command to clear the REPL history and then #load the .csx again. After that, I'll try a new workflow—add the monkey to the context and then attach the country to the monkey that's already being tracked. Note that I'm not including all of the responses in the following listing:

```
> :reset
> #load "SetupForMonkeyTests.csx"
> ctx.Monkeys.Add(monkey);
{...response...}
> monkey.CountryOfOrigin=country;
{...response...}
> ctx.Entry(monkey).State.ToString()
Added
> ctx.Entry(country).State.ToString()
Added
```

Again, the context assigns the Added state to the country object because I attached it to another Added entity.

If you really want to use the navigation property, the pattern that will give you success is to ensure that the context already is aware of the country. This can happen either because you've retrieved the country using a query in the same context instance, resulting in the context assigning the Unchanged state to the object, or because you manually assigned the Unchanged state yourself.

Figure 6 shows an example of the latter, which allows me to do this testing without working with a database.

Because the context was already tracking the country, the country object's state won't be changed. EF only changes the state of the related object when its state is unknown.

Scriptcs Will Be More Than Just a Great Teaching Tool for Me

Personally, I prefer to avoid the confusion around these rules completely and simply set the foreign key value (CountryOfOriginId=country.Id), without attaching a reference using a navigation property. In fact, with that pattern, I can then look more closely at the CountryOfOrigin navigation property and consider if I even want it there. Demonstrating all of the variations and how EF responds to each scenario is an eye-opening lesson for many who have seen this.

When trying to show developers these behaviors, I like the immediate and obvious response I can get from an interactive window. While LINQPad can also help me achieve this, I like the command-line interaction. More important, having used these experiments as a way to introduce myself to Scriptcs, I'm now more aware of the great value it brings beyond just performing command-line tests on an API. ■

Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010), as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Justin Rusbatch*

NEW
Entity Framework Profiler 3.0

Don't waste nights optimizing your database queries.

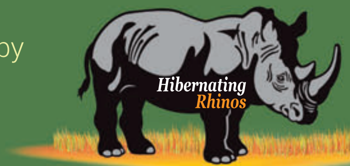
Don't let database dictate how your application works.

Find out how it can help you!

- OPTIMIZE QUERIES
- FIND BOTTLENECKS
- IMPROVE ENTITY FRAMEWORK PERFORMANCE
- SAVE COUNTLESS TIME

www.**hibernatingrhinos**.com

Proudly Developed by



Analyzing Architecture with Code Maps in Visual Studio 2015

Stuart Kent, Jean-Marc Prieur and Blair McGlashan

Improving the architecture of your application is crucial for preventing the build-up of technical debt, and for maintaining good coding velocity. Further, getting a quick understanding of the impact of a potential code change is an important aspect of deciding whether the change makes sense, and if so, what its likely cost will be. Both goals require an ability to understand the architecture of your application and analyze the dependencies surfaced there. To tackle the first goal, you typically work from the top and drill down. To tackle the second, you generally work from a specific code element and expand up.

In this article, we'll show how Code Map, enhanced with new capabilities in Visual Studio 2015 Enterprise, can be used to:

- Understand the overall architecture of a .NET application.
- Analyze dependencies surfaced in that architecture by progressively drilling into the details.
- Understand and analyze the impact of a proposed change to the code by building a dependency map from a specific code element.

This article discusses:

- New capabilities of Code Map in Visual Studio 2015
- Using code maps to understand the overall architecture of an application
- Analyzing dependencies
- Analyzing the impact of a potential change

Technologies discussed:

Visual Studio 2015, Microsoft .NET Framework, Code Map

Improving the architecture of your application is crucial for preventing the build-up of technical debt, and for maintaining good coding velocity.

We used two examples for illustration. For top-down architecture and drilling in, we used the "Roslyn" code base (the .NET Compiler Platform). We chose this for two reasons. First, it's large so understanding the overall architecture isn't so easy. (In fact, it's so large you typically have to wait for a few minutes from a cold start to create the initial diagram, while the initial build and indexing take place. Performance is much better with smaller solutions.)

Second, Roslyn is open source, so you can easily try this for yourself. To analyze the impact of a proposed change, we used an experimental prototype from Microsoft for visualizing a backlog in Team Foundation Server (TFS). This isn't publically available, but we also identify a code element in the Roslyn code base you can explore in a similar way, though the reason for doing so in that case is less clear.

We used the RTM version of Visual Studio 2015 Enterprise edition, which can be downloaded from bit.ly/1JbLA4S.

Understand Overall Architecture

Our starting point was the Roslyn solution opened in Visual Studio, and already built. To get there, we cloned the Git repository from https://github.com/dotnet/roslyn, using the Git cloning experience on the Connect tab of the Team Explorer window of Visual Studio, and then opened the RoslynLight.sln solution, which was automatically found on opening the cloned repository. Then we ran the src\.nuget\NuGetRestore.ps1 script from the Developer command prompt for Visual Studio 2015 to restore the NuGet packages required to build the solution. (Execution of Windows PowerShell scripts is disabled by default. Enabling script execution requires an elevated command prompt. If you're not familiar with Windows PowerShell, you might find it easier to run the script by right-clicking and choosing Run with PowerShell from Windows Explorer.)

Next, we added the xunit.runner.visualstudio NuGet package to the solution so that the Roslyn solution, which are xUnit tests, were detected by Visual Studio. Finally, we built the solution (this can take some time). To get an immediate understanding of the architecture of this code base, we selected the Generate Code Map for Solution without Building

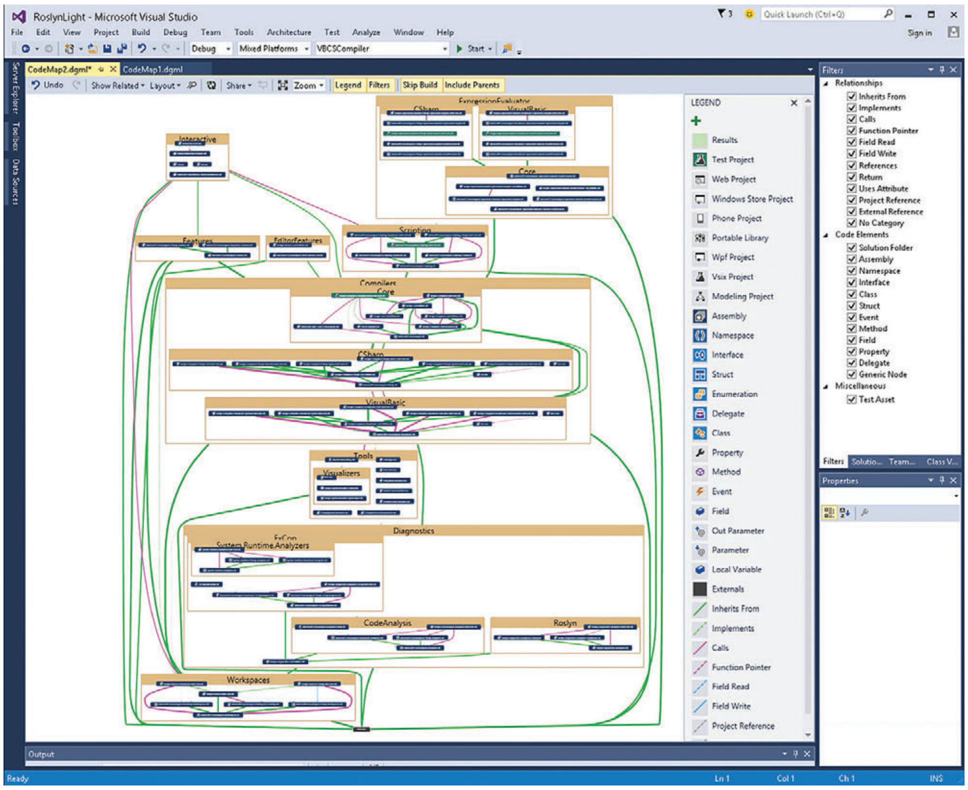


Figure 1 **Architecture Map After Indexing**

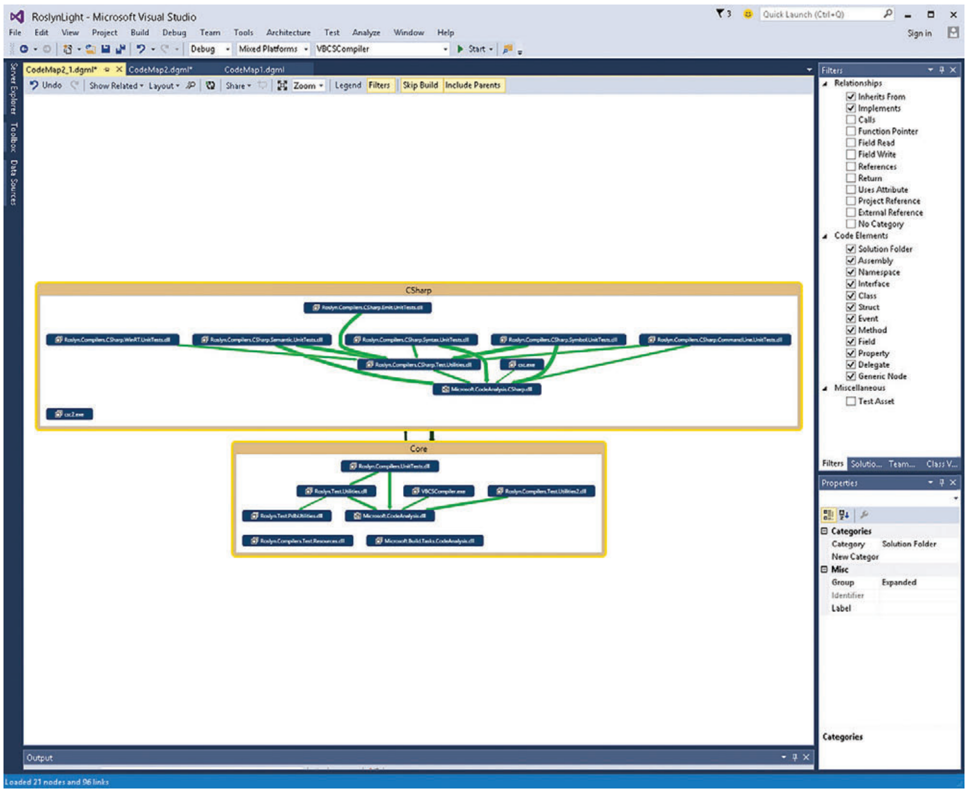
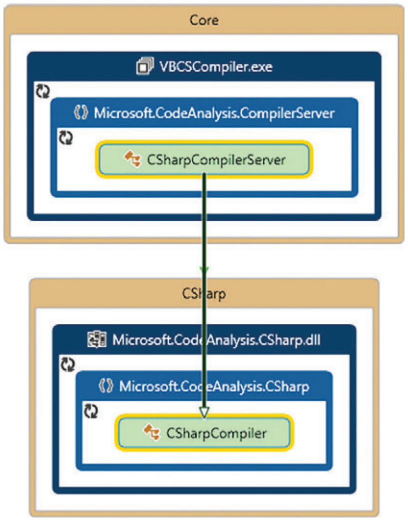


Figure 2 **Zooming in on the Inheritance Hierarchy**

option from the Architecture menu. This created a map that shows a graph of the solution structure and project references. In particular, seeing the references helps you understand how the various parts of the solution are related in a way you simply can't appreciate by looking at the solution, and would take a lot of exploration of project references to figure out.

Once all the binaries have been indexed into the code index (essentially a SQL database storing the full logical structure of the code), the map transforms into what you see in **Figure 1**. The new map displays much richer dependencies, color-coded according to the Legend (click the toggle button in the Code Map toolbar to view), whose thickness reflects the degree of dependency between the related components. For example, a green link indicates inheritance (and possibly other dependencies) between classes in the related projects. The darker-green nodes represent test projects.



Core
VBCSCompiler.exe
Microsoft.CodeAnalysis.CompilerServer
CSharpCompilerServer
CSharp
Microsoft.CodeAnalysis.CSharp.dll
Microsoft.CodeAnalysis.CSharp
CSharpCompiler

Figure 3 **Result of Drilling into the Link Between Core and CSharp**

Not only can you use the map to navigate the code—double-clicking on a node brings you to the correct place in the code—you can also use it to see patterns in the design.

The filters on the right can be used to exclude different kinds of dependency. For example, you might want to view the diagram with all test code hidden, or with all links hidden except inheritance and implements links. You could also hide generic nodes and delete any node containing referenced assemblies that are external to the solution. This makes it easy to see where inheritance relationships exist across the architecture of the product code (tests are not counted as product code).

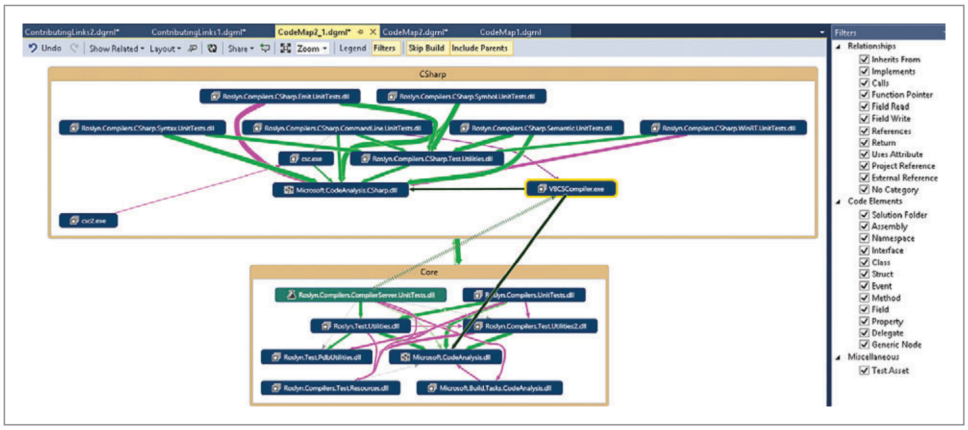
For example, looking at the roots of the Inherits From and Implements relationships gives you a sense of the conceptual abstractions in the framework. You can see that the Compilers framework is at the base of everything else. Within the compilers framework, the Core is also at the base, although there are also some classes in the Core that seem to derive from the CSharp and VisualBasic layers, which is, perhaps, a hint that the layering isn't quite right. (We'll show how to investigate this further in the next section.)

You can zoom in on areas of the diagram to explore further, and even select a node or nodes (control-click enables a multiple selection) to explore on another diagram using the New Graph from Selection context menu command. For example, **Figure 2** illustrates the result of doing this for the Core and CSharp subgroups of the Compilers part of the framework. Bottom-to-top layout has also been applied to the result in order to have base types at the top.

Analyze a Dependency

As you can clearly see in **Figure 2**, the Core inherits some classes from CSharp and VisualBasic, which is a bit of a surprise. Let's take a look at how to drill into this further to see what's causing the anomaly. We started by selecting the dependency link (between Core and CSharp) and choosing Show Contributing Links on New Code Map. The result is shown in **Figure 3**.

We saw that one class is at fault, and had to wonder whether, in fact, the VBCSCompiler.exe project/assembly should really be part of the Core grouping. Perhaps we needed to refactor the Solution Folder structure to move it out. But before making that change, we used Code Map to explore the potential impact. Going back to the map shown in **Figure 2**, we could re-enable all the link filters in order to see all assemblies, and edit the diagram by moving the VBCSCompiler.exe node from Core to CSharp, to see the impact on dependencies (see **Figure 4**). This certainly appeared to clean things up, though we then found that the Roslyn.Compilers.CompilerServer.UnitTests.dll also seemed to be in the wrong place. Exploration can continue in this vein. You can even create new group nodes on the diagram to produce a cleaner architecture, which can then be used to inform



ContributingLinks2.dgml*
ContributingLinks1.dgml*
CodeMap2_1.dgml*
CodeMap2.dgml*
CodeMap1.dgml
Undo
Show Related
Layout
Share
Zoom
Legend
Filters
Skip Build
Include Parents
CSharp
Roslyn.Compilers.CSharp.Emit.UnitTests.dll
Roslyn.Compilers.CSharp.Symbol.UnitTests.dll
Roslyn.Compilers.CSharp.Syntax.UnitTests.dll
Roslyn.Compilers.CSharp.CommandLine.UnitTests.dll
Roslyn.Compilers.CSharp.Semantic.UnitTests.dll
Roslyn.Compilers.CSharp.WinRT.UnitTests.dll
csc.exe
Roslyn.Compilers.CSharp.Test.Utilities.dll
Microsoft.CodeAnalysis.CSharp.dll
VBCSCompiler.exe
csc2.exe
Core
Roslyn.Compilers.CompilerServer.UnitTests.dll
Roslyn.Compilers.UnitTests.dll
Roslyn.Test.Utilities.dll
Roslyn.Compilers.Test.Utilities2.dll
Roslyn.Test.PdbUtilities.dll
Microsoft.CodeAnalysis.dll
Roslyn.Compilers.Test.Resources.dll
Microsoft.Build.Tasks.CodeAnalysis.dll

Filters
- Relationships
  - ✓ Inherits From
  - ✓ Implements
  - ✓ Calls
  - ✓ Function Pointer
  - ✓ Field Read
  - ✓ Field Write
  - ✓ References
  - ✓ Return
  - ✓ Uses Attribute
  - ✓ Project Reference
  - ✓ External Reference
  - ✓ No Category
- Code Elements
  - ✓ Solution Folder
  - ✓ Assembly
  - ✓ Namespace
  - ✓ Interface
  - ✓ Class
  - ✓ Struct
  - ✓ Event
  - ✓ Method
  - ✓ Field
  - ✓ Property
  - ✓ Delegate
  - ✓ Generic Node
- Miscellaneous
  - ✓ Test Asset

Figure 4 **Moving Nodes to Explore the Impact of Potential Refactorings on Dependencies**

refactoring of solution folders, as well as deeper refactorings, such as splitting classes or interfaces between assemblies.

Analyze the Impact of a Proposed Change

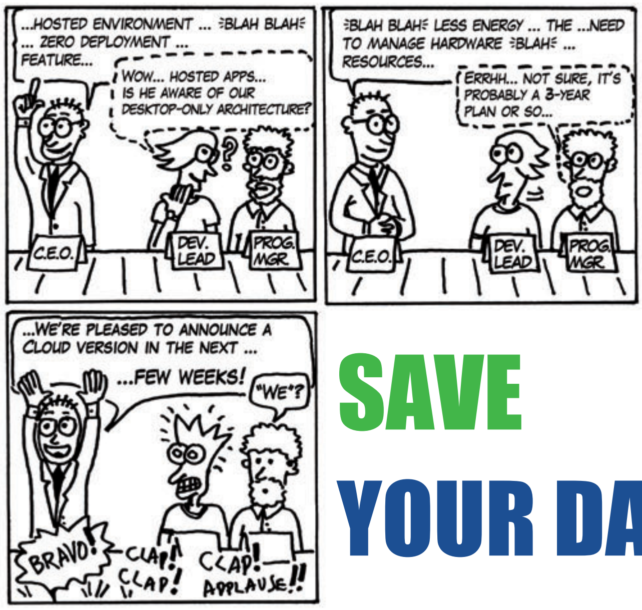
You've seen how to examine the architecture and dependencies of a code base from the top down, and how that can help you identify dependencies to analyze further. You also saw how Code Map can be used effectively to do that analysis by progressively drilling into lower levels of detail. Now we're going to change tack and look at how the same capabilities can be used to explore dependencies with a bottom-up approach, starting from a code element. In particular, we'll identify the set of dependencies that reference that code element and might be impacted when the code element is changed.

Our example shows code from an experimental prototype from Microsoft that's used to visualize and make changes to a backlog in Visual Studio Online or TFS using hierarchical Kanban boards. In this example, we wanted to add the ability to change the title of a work item directly through the map, which requires the addition of a new enumeration literal to the ChangedProperty enum. (If you want to try this with an Enum in the Roslyn code base, you can use Microsoft.Code-Analysis.LocationKind.) First, however, we wanted to explore the impact of making that change, so we clicked on the reference's CodeLens indicator, and then clicked on the Show on Code Map link.

Code Map can be used effectively to do analysis.

You can see the result in **Figure 5**, which presents all the methods and other class members that reference the Changed-Property enum in some way. This diagram shows the members in architectural context; that is, grouped by class, namespace, assembly and solution folder. We then used the filters to eliminate Solution Folders and Assemblies and changed the Layout to a left-to-right format.

This map can now be used to explore all the areas of the code that might be impacted by this change. Not only can you use the map to navigate the code—double-clicking on a node brings you to the correct place in the code—you can also use it to see patterns in the design. So, as **Figure 5** shows, you can immediately see that the method SortAndReparent appears in classes of the StoryMaps.ViewModel namespace, and a quick inspection of the code of any of those classes reveals that no code change is required. However, when you look at the Process method in the WorkItemsWriter class, you can see immediately from the diagram that it calls out to sub-processes, one for each literal in the ChangedProperty enum. Adding a new literal is likely to mean that


...HOSTED ENVIRONMENT ... ≡BLAH BLAH≡ ... ZERO DEPLOYMENT ... FEATURE...
WOW... HOSTED APPS... IS HE AWARE OF OUR DESKTOP-ONLY ARCHITECTURE?
≡BLAH BLAH≡ LESS ENERGY ... THE ...NEED TO MANAGE HARDWARE ≡BLAH≡ ... RESOURCES...
ERRHH... NOT SURE, IT'S PROBABLY A 3-YEAR PLAN OR SO...
...WE'RE PLEASED TO ANNOUNCE A CLOUD VERSION IN THE NEXT ...
...FEW WEEKS!
"WE"?
C.E.O.
DEV. LEAD
PROG. MGR
BRAVO!
CLAP!
CLAP!
CLAP!
APPLAUSE!!

SAVE YOUR DAY!

Use CodeFluent Entities

CodeFluent Entities is a unique product integrated into Visual Studio that allows you to generate database scripts, code (C#, VB), web services and UIs.

*"CodeFluent Entities is a masterpiece software product envisioned and implemented by a group of very talented and experienced people. All that is achieved upon delivering high quality and standardized code and database layers, neatly stitched together and working in unison, lifting the burden from the developers to worry about this aspect on the development process, which could be called plumbing."**

Renato Xavier - Colombaroli - Brazil

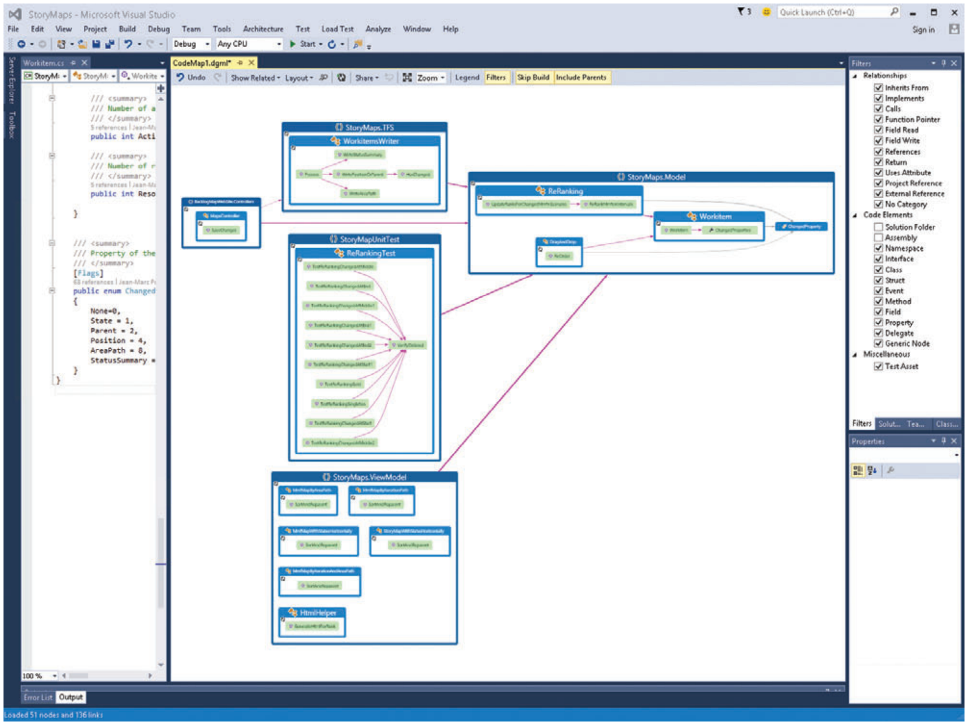
* Source : http://visualstudiogallery.msdn.microsoft.com/B6299BBF-1EF1-436D-B618-66E8C16AB410

To get a license worth $399 for free
Go to www.softfluent.com/forms/msdn-2015

CodeFluent Entities
tools for developers, by developers

More information: www.softfluent.com **Contact us:** info@softfluent.com a new sub-process method will be required. Inspection of the code confirms this suspicion.

As you go through this process, you can make nodes green, red or some other color to indicate whether follow-up is required (use the Edit submenu of the context menu | Flag for Follow-up), as well as add comments to the diagram with more detail (see **Figure 6**).



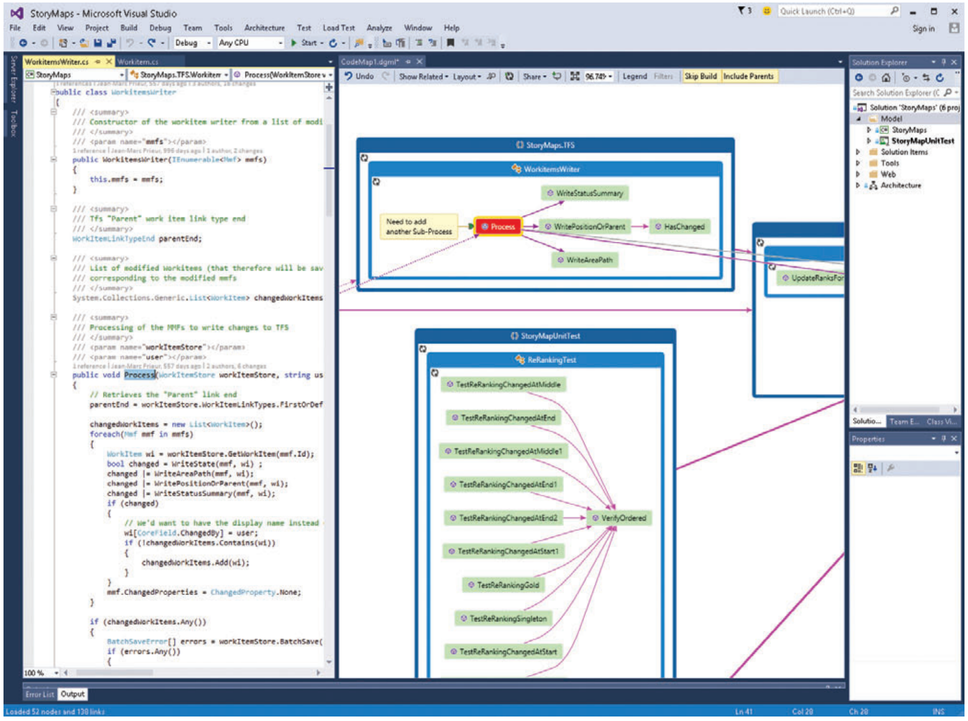
```
/// <summary>
/// Number of a
/// </summary>
public int Acti

/// <summary>
/// Number of r
/// </summary>
public int Reso

}

/// <summary>
/// Property of the
/// </summary>
[Flags]
public enum Changed
{
    None=0,
    State = 1,
    Parent = 2,
    Position = 4,
    AreaPath = 8,
    StatusSummary =
}
}
```

Figure 5 **The Reference Dependency Map After Hiding Solution Folders and Assemblies and Laying out Left to Right**



```
public class WorkitemsWriter
{
    /// <summary>
    /// Constructor of the workitem writer from a list of modi
    /// </summary>
    /// <param name="mmfs"></param>
    public WorkitemsWriter(IEnumerable<Mmf> mmfs)
    {
        this.mmfs = mmfs;
    }

    /// <summary>
    /// Tfs "Parent" work item link type end
    /// </summary>
    WorkItemLinkTypeEnd parentEnd;

    /// <summary>
    /// List of modified Workitems (that therefore will be sav
    /// corresponding to the modified mmfs
    /// </summary>
    System.Collections.Generic.List<WorkItem> changedWorkItems

    /// <summary>
    /// Processing of the MMFs to write changes to TFS
    /// </summary>
    /// <param name="workItemStore"></param>
    /// <param name="user"></param>
    public void Process(WorkItemStore workItemStore, string us
    {
        // Retrieves the "Parent" link end
        parentEnd = workItemStore.WorkItemLinkTypes.FirstOrDef

        changedWorkItems = new List<WorkItem>();
        foreach(Mmf mmf in mmfs)
        {
            WorkItem wi = workItemStore.GetWorkItem(mmf.Id);
            bool changed = WriteState(mmf, wi) ;
            changed |= WriteAreaPath(mmf, wi);
            changed |= WritePositionOrParent(mmf, wi);
            changed |= WriteStatusSummary(mmf, wi);
            if (changed)
            {
                // We'd want to have the display name instead
                wi[CoreField.ChangedBy] = user;
                if (!changedWorkItems.Contains(wi))
                {
                    changedWorkItems.Add(wi);
                }
            }
            mmf.ChangedProperties = ChangedProperty.None;
        }

        if (changedWorkItems.Any())
        {
            BatchSaveError[] errors = workItemStore.BatchSave(
            if (errors.Any())
            {
```

StoryMaps - Microsoft Visual Studio
StoryMaps.TFS
WorkitemsWriter
WriteStatusSummary
Need to add another Sub-Process
Process
WritePositionOrParent
HasChanged
WriteAreaPath
UpdateRanksFor
StoryMapUnitTest
ReRankingTest
TestReRankingChangedAtMiddle
TestReRankingChangedAtEnd
TestReRankingChangedAtMiddle1
TestReRankingChangedAtEnd1
TestReRankingChangedAtEnd2
VerifyOrdered
TestReRankingChangedAtStart1
TestReRankingGold
TestReRankingSingleton
TestReRankingChangedAtStart
Loaded 52 nodes and 130 links

Figure 6 **Annotated References Dependency Map**

Wrapping Up

Understanding the architecture of your application is important, and so is knowing the impact of any potential code change.

In this article, we showed how to use Code Map, enhanced with new capabilities in Visual Studio 2015, to understand and analyze the overall architecture of your .NET app, and examine dependencies at a high level. These capabilities include:

- A much-improved top-level diagram generated from a solution, which uses solution folders to provide an initial grouping of nodes, and styles project nodes according to their type.
- The ability to create new code maps from a selection on an existing map.
- Node and link filtering, in particular the ability to filter out tests and different kinds of links.

We then showed how drilling into a dependency link, enhanced in Visual Studio 2015 to make use of link filters, can quickly unearth the cause of an unwanted dependency, right down to the line of code where the dependency is introduced.

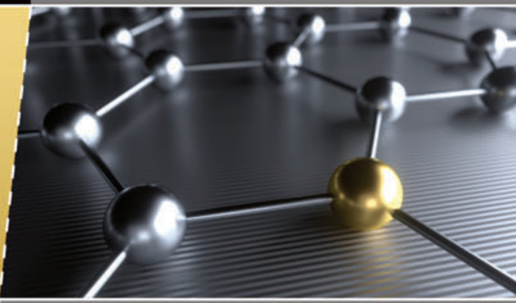
Finally, we showed how you can take a bottom-up approach with code maps to reveal the impact of a proposed code change by exploring the dependency map created from the reference's CodeLens indicator from a specific code element, and then using comments and flags in the diagram to record the results of analyzing the impact of making the change. ■

Stuart Kent *is a group program manager at Microsoft responsible for developer experiences in Visual Studio and Visual Studio Online, focused on controlling technical debt and code sharing and collaboration. This includes architecture analysis tools, aspects of CodeLens and code search.*

Jean-Marc Prieur *is a senior program manager at Microsoft envisioning and driving the delivery of experiences in Visual Studio and Visual Studio Online focused on controlling technical debt, including architecture analysis tools.*

Blair McGlashan *is an engineering manager leading a team based in Cambridge, U.K., delivering experiences focused on technical debt, including architecture analysis tools.*

Switch to Amyuni PDF



Create & Edit PDFs in .Net - ActiveX - WinRT

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .Net and ActiveX/COM
- New WinRT Component enables publishing C#, C++CX or Javascript apps to Windows Store
- New Postscript/EPS to PDF conversion module

Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as JPeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment

Advanced HTML to PDF & XAML

- Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- Easy Integration and deployment within developer's applications
- WebkitPDF is based on the Webkit Open Source library and Amyuni PDF Creator

High Performance PDF Printer For Desktops and Servers

- Our high-performance printer driver optimized for Web, Application and Print Servers. Print to PDF in a fraction of the time needed with other tools. WHQL tested for Windows 32 and 64-bit including Windows Server 2012 R2 and Windows 8.1
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language
- Easy licensing and deployment to fit system administrator's requirements

CERTIFIED FOR Windows Server 2008 · Windows 7 · Windows 8 · Windows Server 2012 Certified

AMYUNI

All development tools available at

www.amyuni.com

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.

Brownfield Async Development

Stephen Cleary

When the Visual Studio Async CTP came out, I was in a fortunate position. I was the sole developer for two relatively small greenfield applications that would benefit from async and await. During this time, various members of the MSDN forums including myself were discovering, discussing and implementing several asynchronous best practices. The most important of those practices are compiled into my March 2013 *MSDN Magazine* article, "Best Practices in Asynchronous Programming" (msdn.microsoft.com/magazine/jj991977).

Applying async and await to an existing code base is a different kind of challenge. Brownfield code can be messy, which further complicates the scenario. A few techniques I've found useful when applying async to brownfield code I'll explain here. Introducing async can actually affect the design in some cases. If there's any refactoring necessary to separate the existing code into layers, I recommend doing that before introducing async. For the purposes of this article, I'll assume you're using an application architecture similar to what's shown in **Figure 1**.

This article discusses:

- Convert synchronous code to asynchronous
- Resolving deadlocks in code operations
- Applying hacks to get around code conflicts

Technologies discussed:

Visual Studio Async CTP, C#, Microsoft .NET Framework

When to Use Async

The best general approach is to first think about what the application is actually doing. Async excels at I/O-bound operations, but there are sometimes better options for other kinds of processing. There are two somewhat common scenarios where async isn't a perfect fit—CPU-bound code and data streams.

If you have CPU-bound code, consider the Parallel class or Parallel LINQ. Async is more suited to an event-based system, where there's no actual code executing while an operation is in progress. CPU-bound code within an async method will still run synchronously.

However, you can treat CPU-bound code as though it were asynchronous by awaiting the result of Task.Run. This is a good way to push CPU-bound work off the UI thread. The following code is an example of using Task.Run as a bridge between asynchronous and parallel code:

```
await Task.Run(() => Parallel.ForEach(...));
```

The other scenario where async isn't the best fit is when your application is dealing with data streams. Async operations have a definite beginning and end. For example, a resource download starts when the resource is requested. It finishes when the resource download completes. If your incoming data is more of a stream or subscription, then async may not be the best approach. Consider a device attached to a serial port that may volunteer data at any time, as an example.

It's possible to use async/await with event streams. It will require some system resources for buffering the data as it arrives until the application reads the data. If your source is an event subscription, consider using Reactive Extensions or TPL Dataflow. You might

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Redmond AUGUST 10 - 14
MICROSOFT HEADQUARTERS, REDMOND, WA

NAVIGATE THE .NET HIGHWAY

CODE HOME

5 DAYS OF EDUCATIONAL SESSIONS AND WORKSHOPS

UNDER CONSTRUCTION

KEYNOTE AND MICROSOFT SESSION DETAILS COMING SOON!

With all of the announcements coming out of Build and Ignite, we'll be finalizing the Redmond Keynotes and the **FULL Track** of Microsoft-led sessions shortly – go to **vslive.com/redmond** for content updates!

REGISTER BY JULY 8 AND SAVE $300

Scan the QR code to register or for more event details. Use promo code REDJULTI

EVENT SPONSOR
Microsoft

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA INC

vslive.com/redmond

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Redmond AUGUST 10 - 14
MICROSOFT HEADQUARTERS, REDMOND, WA

NAVIGATE THE .NET HIGHWAY

Development Tracks include:

- Visual Studio/.NET
- Web Development
- Design
- Mobile Client
- Windows Client
- Database and Analytics
- Cloud Computing
- Microsoft Sessions

REGISTER BY JULY 8 AND SAVE $300

Scan the QR code to register or for more event details.
Use promo code REDJULTI

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive
facebook.com – Search "VSLive"
linkedin.com – Join the "Visual Studio Live" group!

find it a more natural fit than plain async. Both Rx and Dataflow interoperate nicely with asynchronous code.

Async is certainly the best approach for quite a lot of code, just not all of it. For the remainder of this article, I'll assume you've considered the Task Parallel Library and Rx/Dataflow and have concluded that async/await is the most appropriate approach.

Transform Synchronous to Asynchronous Code

There's a regular procedure for converting existing synchronous code into asynchronous code. It's fairly straightforward. It may even become rather tedious once you've done it a few times. As of this writing, there's no support for automatic synchronous-to-asynchronous conversion. However, I expect this kind of code transformation will be introduced in the next few years.

This procedure works best when you start at the lower-level layers and work your way toward the user levels. In other words, start introducing async in the data layer methods that access a database or Web APIs. Then introduce async in your service methods, then the business logic and, finally, the user layer. If your code doesn't have well-defined layers, you can still convert to async/await. It will just be a bit more difficult.

The first step is to identify the low-level naturally asynchronous operation to convert. Anything I/O-based is a prime candidate for async. Common examples are database queries and commands, Web API calls and file system access. Many times, this low-level operation already has an existing asynchronous API.

If the underlying library has an async-ready API, all you need to do is add an Async suffix (or TaskAsync suffix) on the synchronous method name. For example, an Entity Framework call to First can be replaced with a call to FirstAsync. In some cases, you might want to use an alternative type. For example, HttpClient is a more async-friendly replacement for WebClient and HttpWebRequest. In some cases, you might need to upgrade the version of your library. Entity Framework, for example, acquired an async API in version 6.

Consider the code in **Figure 1**. This is a simple example with a service layer and some business logic. In this example, there's only one low-level operation—retrieving a frob identifier string from a Web API in WebDataService.Get. This is the logical place to begin the asynchronous conversion. In this case, the developer can choose to either replace WebClient.DownloadString with WebClient.DownloadStringTaskAsync, or replace WebClient with the more async-friendly HttpClient.

The second step is to change the synchronous API call to an asynchronous API call, and then await the returned task. When code invokes an asynchronous method, it's generally proper to await the returned task. At this point, the compiler will complain. The following code will cause a compiler error with the message, "The 'await' operator can only be used within an async method. Consider marking this method with the 'async' modifier and changing its return type to 'Task<string>'":

```
public sealed class WebDataService : IDataService
{
  public string Get(int id)
  {
    using (var client = new WebClient())
      return await client.DownloadStringTaskAsync(
      "http://www.example.com/api/values/" + id);
  }
}
```

The compiler guides you to the next step. Mark the method as async and change the return type. If the return type of the synchronous method is void, then the return type of the asynchronous

Figure 1 **Simple Code Structure with a Service Layer and Business Logic Layer**

```
public interface IDataService
{
  string Get(int id);
}

public sealed class WebDataService : IDataService
{
  public string Get(int id)
  {
    using (var client = new WebClient())
      return client.DownloadString("http://www.example.com/api/values/" + id);
  }
}

public sealed class BusinessLogic
{
  private readonly IDataService _dataService;

  public BusinessLogic(IDataService dataService)
  {
    _dataService = dataService;
  }

  public string GetFrob()
  {
    // Try to get the new frob id.
    var result = _dataService.Get(17);
    if (result != string.Empty)
      return result;

    // If the new one isn't defined, get the old one.
    return _dataService.Get(13);
  }
}
```

Figure 2 **Change All Calling Methods to Async**

```
public interface IDataService
{
  Task<string> GetAsync(int id);
}

public sealed class WebDataService : IDataService
{
  public async Task<string> GetAsync(int id)
  {
    using (var client = new WebClient())
      return await client.DownloadStringTaskAsync(
      "http://www.example.com/api/values/" + id);
  }
}

public sealed class BusinessLogic
{
  private readonly IDataService _dataService;

  public BusinessLogic(IDataService dataService)
  {
    _dataService = dataService;
  }

  public async Task<string> GetFrobAsync()
  {
    // Try to get the new frob id.
    var result = await _dataService.GetAsync(17);
    if (result != string.Empty)
      return result;

    // If the new one isn't defined, get the old one.
    return await _dataService.GetAsync(13);
  }
}
```

method should be Task. Otherwise, for any synchronous method return type of T, the asynchronous method return type should be Task<T>. When you change the return type to Task/Task<T>, you should also modify the method name to end in Async, to follow the Task-Based Asynchronous Pattern guidelines. The following code shows the resulting method as an asynchronous method:

```
public sealed class WebDataService : IDataService
{
  public async Task<string> GetAsync(int id)
  {
    using (var client = new WebClient())
      return await client.DownloadStringTaskAsync(
      "http://www.example.com/api/values/" + id);
  }
}
```

Before moving on, check the rest of this method for any other blocking or synchronous API calls that you can make async. Asynchronous methods shouldn't block, so this method should call asynchronous APIs if they're available. In this simple example, there are no other blocking calls. In real-world code, keep an eye out for retry logic and optimistic conflict resolution.

Entity Framework should get a special mention here. One subtle "gotcha" is lazy loading of related entities. This is always done synchronously. If possible, use additional explicit asynchronous queries instead of lazy loading.

Now this method is finally done. Next, move to all methods that reference this one, and follow this procedure again. In this case, WebDataService.Get was part of an interface implementation, so you must change the interface to enable asynchronous implementations:

```
public interface IDataService
{
  Task<string> GetAsync(int id);
}
```

Next, move to the calling methods and follow the same steps. You should end up with something like the code in **Figure 2**. Unfortunately, code won't compile until all calling methods are transformed to async, and then all of their calling methods are transformed to async, and so on. This cascading nature of async is the burdensome aspect of brownfield development.

Eventually, the level of asynchronous operation in your code base will grow until it hits a method that isn't called by any other methods in your code. Your top-level methods are called directly by whichever framework you're using. Some frameworks such as ASP.NET MVC permit asynchronous code directly. For example, ASP.NET MVC controller actions can return Task or Task<T>. Other frameworks such as Windows Presentation Foundation (WPF) permit asynchronous event handlers. So, for example, a button click event might be async void.

Performing an asynchronous code transformation can be scary the first few times, but it really becomes second nature after a bit of practice.

Figure 3 **Use Vertical Partitions to Convert Sections of Code to Async**

```
public interface IDataService
{
  string Get(int id);
  Task<string> GetAsync(int id);
}

public sealed class WebDataService : IDataService
{
  public string Get(int id)
  {
    using (var client = new WebClient())
      return client.DownloadString("http://www.example.com/api/values/" + id);
  }

  public async Task<string> GetAsync(int id)
  {
    using (var client = new WebClient())
      return await client.DownloadStringTaskAsync(
      "http://www.example.com/api/values/" + id);
  }
}

public sealed class BusinessLogic
{
  private readonly IDataService _dataService;

  public BusinessLogic(IDataService dataService)
  {
    _dataService = dataService;
  }

  public string GetFrob()
  {
    // Try to get the new frob id.
    var result = _dataService.Get(17);
    if (result != string.Empty)
      return result;

    // If the new one isn't defined, get the old one.
    return _dataService.Get(13);
  }

  public async Task<string> GetFrobAsync()
  {
    // Try to get the new frob id.
    var result = await _dataService.GetAsync(17);
    if (result != string.Empty)
      return result;

    // If the new one isn't defined, get the old one.
    return await _dataService.GetAsync(13);
  }
}
```

Hit the Wall

As the level of asynchronous code grows throughout your application, you might reach a point where it seems impossible to continue. The most common examples of this are object-oriented constructs, which don't mesh with the functional nature of asynchronous code. Constructors, events and properties have their own challenges.

Rethinking the design is generally the best way around these difficulties. One common example is constructors. In the synchronous code, a constructor method might block on I/O. In the asynchronous world, one solution is to use an asynchronous factory method instead of a constructor. Another example is properties. If a property synchronously blocks on I/O, that property should probably be a method. An asynchronous conversion exercise is great at exposing these kinds of design issues that creep into your code base over time.

Extreme Performance Linear Scalability

For .NET & Java Apps

(Available on Microsoft Azure & AWS)

Cache data, reduce expensive database trips, and scale your apps to extreme transaction processing (XTP) with NCache.

In-Memory Distributed Cache

- Extremely fast & linearly scalable with 100% uptime
- Mirrored, Replicated, Partitioned, and Client Cache
- Entity Framework & NHibernate Second Level Cache

Open Source

ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider

Full Integration with Microsoft Visual Studio

- NuGet Package for NCache SDK
- Microsoft Certified for Windows Server 2012 R2

NCache™

FREE Download

sales@alachisoft.com US: +1 (925) 236 3830 www.alachisoft.com

Figure 4 **Service Layer Code Using the Blocking Hack**

```
public sealed class WebDataService : IDataService
{
  public string Get(int id)
  {
    return GetAsync(id).GetAwaiter().GetResult();
  }

  public async Task<string> GetAsync(int id)
  {
    // This code will not work as expected.
    using (var client = new WebClient())
      return await client.DownloadStringTaskAsync(
      "http://www.example.com/api/values/" + id);
  }
}
```

Transformation Tips

Performing an asynchronous code transformation can be scary the first few times, but it really becomes second nature after a bit of practice. As you feel more comfortable with converting synchronous code to asynchronous, here are a few tips you can start using during the conversion process.

As you convert your code, keep an eye out for concurrency opportunities. Asynchronous-concurrent code is often shorter and simpler than synchronous-concurrent code. For example, consider a method that has to download two different resources from a REST API. The synchronous version of that method would almost certainly download one and then the other. However, the asynchronous version could easily start both downloads and then asynchronously wait for both to complete using Task.WhenAll.

Another consideration is cancellation. Usually, synchronous application users are used to waiting. If the UI is responsive in the new version, they might expect the ability to cancel the operation. Asynchronous code should generally support cancellation unless there's some other reason it can't. For the most part, your own asynchronous code can support cancellation just by taking a CancellationToken argument and passing it through to the asynchronous methods it calls.

You can convert any code using Thread or BackgroundWorker to use Task.Run instead. Task.Run is far easier to compose than Thread or BackgroundWorker. For example, it's much easier to express, "start two background computations and then do this other thing when they have both completed," with the modern await and Task.Run, than with primitive threading constructs.

Figure 5 **Use the AsyncContext Type**

```
[TestClass]
public class WebDataServiceUnitTests
{
  [TestMethod]
  public async Task GetAsync_RetrievesObject13()
  {
    var service = new WebDataService();
    var result = await service.GetAsync(13);
    Assert.AreEqual("frob", result);
  }

  [TestMethod]
  public void Get_RetrievesObject13()
  {
    AsyncContext.Run(() =>
    {
      var service = new WebDataService();
      var result = service.Get(13);
      Assert.AreEqual("frob", result);
    });
  }
}
```

Vertical Partitions

The approach described so far works great if you're the only developer for your application, and you have no issues or requests that would interfere with your asynchronous conversion work. That's not very realistic, though, is it?

If you don't have the time to convert your entire code base to be asynchronous all at once, you can approach the conversion with a slight modification called vertical partitions. Using this technique, you can do your asynchronous conversion to certain sections of code. Vertical partitions are ideal if you'd like to just "try out" asynchronous code.

To create a vertical partition, identify the user-level code you'd like to make asynchronous. Perhaps it's the event handler for a UI button that saves to a database (where you'd like to keep the UI responsive), or a heavily used ASP.NET request that does the same (where you'd like to reduce the resources required for that specific request). Walk through the code, laying out the call tree for that method. Then you can start at the low-level methods and transform your way up the tree.

Other code will no doubt use those same low-level methods. Because you're not ready to make all that code asynchronous, the solution is to create a copy of the method. Then transform that copy to be asynchronous. This way, the solution can still build at every step. When you've worked your way up to the user-level code, you'll have created a vertical partition of asynchronous code within your application. A vertical partition based on our example code would appear as shown in **Figure 3**.

You may have noticed there's some code duplication with this solution. All the logic for the synchronous and asynchronous methods is duplicated, which isn't good. In a perfect world, code duplication for this vertical partition is only temporary. The duplicated code would exist only in your source control until the application has been completely converted. At this point, you can remove all old synchronous APIs.

However, you can't do this in all situations. If you're developing a library (even one only used internally), backward compatibility is a primary concern. You might find yourself needing to maintain synchronous APIs for quite some time.

Figure 6 **Use HttpClient with ConfigureAwait(false) to Prevent Deadlock**

```
public sealed class WebDataService : IDataService
{
  public string Get(int id)
  {
    return GetAsync(id).GetAwaiter().GetResult();
  }

  public async Task<string> GetAsync(int id)
  {
    using (var client = new HttpClient())
      return await client.GetStringAsync(
      "http://www.example.com/api/values/" + id).ConfigureAwait(false);
  }
}
```

DocuVieware

Universal HTML5 Viewer and Document Management Kit

View, annotate and manage any document, on any device, on any browser through a fully customizable zero-footprint HTML5 Control. Supports 90+ file formats, including PDF, TIFF and SVG.

Why **DocuVieware**?

Cross-platform, any browser, **zero-footprint** solution (no client-side install).

Super-**easy integration** within existing web applications.

Fast and **crystal-clear rendering** of documents and annotations.

Fully **customizable UI** look and feel, including convenient **snap-ins**.

Mobile devices optimization & **responsive** design.

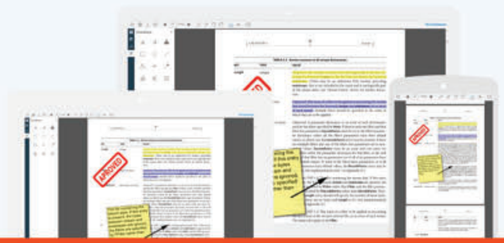
Built-in **annotations**, **thumbnails**, **bookmarks** and **text search** tools.

Works in all modern browsers ...

IE9+, Chrome, Chrome for Android, Firefox, Firefox for Android, Opera, Safari 5+, Mobile Safari.

... so it works on all devices.

PC, Mac, tablets and smartphones.



Powered by GdPicture.NET

www.docuvieware.com

Figure 7 **Code for the Thread Pool Hack**

```
public sealed class WebDataService : IDataService
{
  public string Get(int id)
  {
    return Task.Run(() => GetAsync(id)).GetAwaiter().GetResult();
  }

  public async Task<string> GetAsync(int id)
  {
    using (var client = new WebClient())
      return await client.DownloadStringTaskAsync(
      "http://www.example.com/api/values/" + id);
  }
}
```

Figure 8 **Use a Main Loop for the Thread Pool Hack**

```
public sealed class WebDataService : IDataService
{
  public string Get(int id)
  {
    var task = Task.Run(() => AsyncContext.Run(() => GetAsync(id)));
    return task.GetAwaiter().GetResult();
  }

  public async Task<string> GetAsync(int id)
  {
    using (var client = new WebClient())
      return await client.DownloadStringTaskAsync(
      "http://www.example.com/api/values/" + id);
  }
}
```

There are three possible responses to this situation. First, you could drive adoption of asynchronous APIs. If your library has asynchronous work to do, it should expose asynchronous APIs. Second, you could accept the code duplication as a necessary evil for backward compatibility. This is an acceptable solution only if your team has exceptional self-discipline or if the backward-compatibility constraint is only temporary.

The third solution is to apply one of the hacks outlined here. While I can't really recommend any of these hacks, they can be useful in a pinch. Because their operation is naturally asynchronous, each of these hacks is oriented around providing a synchronous API for a naturally asynchronous operation, which is a well-known anti-pattern described in greater detail in a Server & Tools Blogs post at bit.ly/1JDLmWD.

Figure 9 **Flag Argument Hack Exposes Two APIs**

```
public interface IDataService
{
  string Get(int id);
  Task<string> GetAsync(int id);
}

public sealed class WebDataService : IDataService
{
  private async Task<string> GetCoreAsync(int id, bool sync)
  {
    using (var client = new WebClient())
    {
      return sync
        ? client.DownloadString("http://www.example.com/api/values/" + id)
        : await client.DownloadStringTaskAsync(
        "http://www.example.com/api/values/" + id);
    }
  }

  public string Get(int id)
  {
    return GetCoreAsync(id, sync: true).GetAwaiter().GetResult();
  }

  public Task<string> GetAsync(int id)
  {
    return GetCoreAsync(id, sync: false);
  }
}
```

The Blocking Hack

The most straightforward approach is to simply block the asynchronous version. I recommend blocking with GetAwaiter().GetResult instead of Wait or Result. Wait and Result will wrap any exceptions within an AggregateException, which complicates error handling. The sample service layer code would look like the code shown in **Figure 4** if it used the blocking hack.

Unfortunately, as the comment implies, that code wouldn't actually work. It results in a common deadlock described in my "Best Practices in Asynchronous Programming" article I mentioned earlier.

This is where the hack can get tricky. A normal unit test will pass, but the same code will deadlock if called from a UI or ASP.NET context. If you use the blocking hack, you should write unit tests that check this behavior. The code in **Figure 5** uses the Async-Context type from my AsyncEx library, which creates a context similar to a UI or ASP.NET context.

The asynchronous unit test passes, but the synchronous unit test never completes. This is the classic deadlock problem. The asynchronous code captures the current context and attempts to resume on it, while the synchronous wrapper blocks a thread in that context, preventing the asynchronous operation from completing.

In this case, our asynchronous code is missing a ConfigureAwait-(false). However, the same problem can be caused by using WebClient. WebClient uses the older event-based asynchronous pattern (EAP),

Figure 10 **Apply Flag Argument Hack to Business Logic**

```
public sealed class BusinessLogic
{
  private readonly IDataService _dataService;

  public BusinessLogic(IDataService dataService)
  {
    _dataService = dataService;
  }

  private async Task<string> GetFrobCoreAsync(bool sync)
  {
    // Try to get the new frob id.
    var result = sync
      ? _dataService.Get(17)
      : await _dataService.GetAsync(17);
    if (result != string.Empty)
      return result;

    // If the new one isn't defined, get the old one.
    return sync
      ? _dataService.Get(13)
      : await _dataService.GetAsync(13);
  }

  public string GetFrob()
  {
    return GetFrobCoreAsync(sync: true).GetAwaiter().GetResult();
  }

  public Task<string> GetFrobAsync()
  {
    return GetFrobCoreAsync(sync: false);
  }
}
```

which always captures the context. So even if your code uses ConfigureAwait(false), the same deadlock will occur from the WebClient code. In this case, you can replace WebClient with the more async-friendly HttpClient and get this to work on the desktop, as shown in **Figure 6**.

The blocking hack requires your team to have strict discipline. They need to ensure ConfigureAwait(false) is used everywhere. They must also require all dependent libraries to follow the same discipline. In some cases, this just isn't possible. As of this writing, even HttpClient captures the context on some platforms.

Another drawback to the blocking hack is it requires you to use ConfigureAwait(false). It's simply unsuitable if the asynchronous code actually does need to resume on captured context. If you do adopt the blocking hack, you're strongly recommended to perform unit tests using AsyncContext or another similar single-threaded context to catch any lurking deadlocks.

The Thread Pool Hack

A similar approach to the Blocking Hack is to offload the asynchronous work to the thread pool, then block on the resulting task. The code using this hack would look like the code shown in **Figure 7**.

The call to Task.Run executes the asynchronous method on a thread pool thread. Here it will run without a context, thus avoiding the deadlock. One of the problems with this approach is the asynchronous method can't depend on executing within a specific context. So, it can't use UI elements or the ASP.NET HttpContext.Current.

Another more subtle "gotcha" is the asynchronous method may resume on any thread pool thread. This isn't a problem for most code. It can be a problem if the method uses per-thread state or if it implicitly depends on the synchronization provided by a UI context.

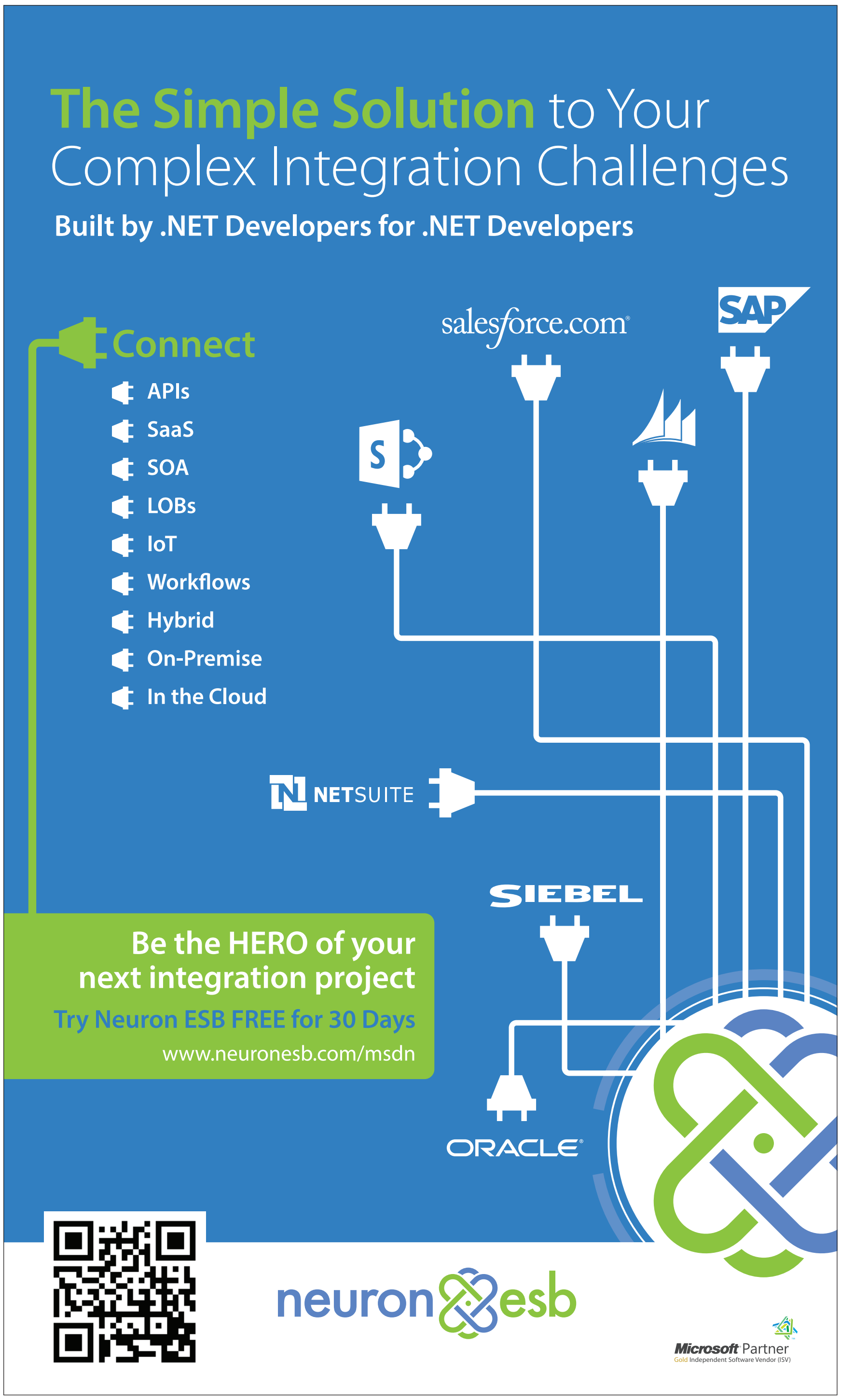
You can create a context for a background thread. The AsyncContext type in my AsyncEx library will install a single-threaded context complete with a "main loop." This forces the asynchronous code to resume on the same thread. This avoids the more subtle "gotchas" of the thread pool hack. The example code with a main loop for the thread pool thread would look like the code shown in **Figure 8**.

Of course, there's a disadvantage to this approach, as well. The thread pool thread is blocked within the AsyncContext until the asynchronous method completes. This blocked thread is there, as well as the primary thread calling the synchronous API. So, for the duration of the call, there are two threads being blocked. On ASP.NET in particular, this approach will significantly reduce the application's ability to scale.

The Flag Argument Hack

This hack is one I haven't used yet. It was described to me by Stephen Toub during his tech review of this article. It's a great approach and my favorite of all these hacks.

The flag argument hack takes the original method, makes it private, and adds a flag to indicate whether the method should run


The Simple Solution to Your Complex Integration Challenges
Built by .NET Developers for .NET Developers
Connect
APIs
SaaS
SOA
LOBs
IoT
Workflows
Hybrid
On-Premise
In the Cloud
salesforce.com
SAP
NETSUITE
SIEBEL
ORACLE
Be the HERO of your next integration project
Try Neuron ESB FREE for 30 Days
www.neuronesb.com/msdn
neuron esb
Microsoft Partner
Gold Independent Software Vendor (ISV)

synchronously or asynchronously. It then exposes two public APIs, one synchronous and the other asynchronous, as shown in **Figure 9**.

The GetCoreAsync method in this example has one important property—if its sync argument is true, it always returns an already-completed task. The method will block when its flag argument requests synchronous behavior. Otherwise, it acts just like a normal asynchronous method.

The synchronous Get wrapper passes true for the flag argument and then retrieves the result of the operation. Note there's no chance of a deadlock because the task is already completed. The business logic follows a similar pattern, as shown in **Figure 10**.

You do have the option of exposing the CoreAsync methods from your service layer. This simplifies the business logic. However, the flag argument method is more of an implementation detail. You'd need to weigh the advantage of cleaner code against the disadvantage of exposing implementation details, as shown in **Figure 11**. The advantage of this hack is the logic of the methods stays basically the same. It just calls different APIs based on the value of the flag argument. This works great if there's a one-to-one correspondence between synchronous and asynchronous APIs, which is usually the case.

It may not work as well if you want to add concurrency to your asynchronous code path, or if there's not an ideal corresponding asynchronous API. For example, I would prefer to use HttpClient over WebClient in WebDataService, but I'd have to weigh that against the added complexity it would cause in the GetCoreAsync method.

The primary disadvantage of this hack is flag arguments are a well-known anti-pattern. Boolean flag arguments are a good indicator a method is really two different methods in one. However, the anti-pattern is minimized within the implementation details of a single class (unless you choose to expose your CoreAsync methods). Despite this, it's still my favorite of the hacks.

Figure 11 **Implementation Details Are Exposed, but the Code Is Clean**

```
public interface IDataService
{
  string Get(int id);
  Task<string> GetAsync(int id);
  Task<string> GetCoreAsync(int id, bool sync);
}

public sealed class BusinessLogic
{
  private readonly IDataService _dataService;

  public BusinessLogic(IDataService dataService)
  {
    _dataService = dataService;
  }

  private async Task<string> GetFrobCoreAsync(bool sync)
  {
    // Try to get the new frob id.
    var result = await _dataService.GetCoreAsync(17, sync);
    if (result != string.Empty)
      return result;

    // If the new one isn't defined, get the old one.
    return await _dataService.GetCoreAsync(13, sync);
  }

  public string GetFrob()
  {
    return GetFrobCoreAsync(sync: true).GetAwaiter().GetResult();
  }

  public Task<string> GetFrobAsync()
  {
    return GetFrobCoreAsync(sync: false);
  }
}
```

Figure 12 **Execute a Nested Message with AsyncContext**

```
public sealed class WebDataService : IDataService
{
  public string Get(int id)
  {
    return AsyncContext.Run(() => GetAsync(id));
  }

  public async Task<string> GetAsync(int id)
  {
    using (var client = new WebClient())
      return await client.DownloadStringTaskAsync(
      "http://www.example.com/api/values/" + id);
  }
}
```

The Nested Message Loop Hack

This final hack is my least favorite. The idea is that you set up a nested message loop within the UI thread and execute the asynchronous code within that loop. This approach isn't an option on ASP.NET. It may also require different code for various UI platforms. For example, a WPF application could use nested dispatcher frames, while a Windows Forms application could use DoEvents within a loop. If the asynchronous methods don't depend on a particular UI platform, you can also use AsyncContext to execute a nested loop, as shown in **Figure 12**.

Don't be deceived by the simplicity of this example code. This hack is the most dangerous of them all, because you must consider reentrancy. This is particularly true if the code uses nested dispatcher frames or DoEvents. In that case, the entire UI layer must now handle unexpected reentrancy. Reentrant-safe applications require a considerable amount of careful thought and planning.

Wrapping Up

In an ideal world, you could perform a relatively simple code transformation from synchronous to asynchronous and everything would be rainbows and unicorns. In the real world, it's often necessary for synchronous and asynchronous code to coexist. If you just want to try out async, create a vertical partition (with code duplication) until you're comfortable using async. If you must maintain the synchronous code for backward-compatibility reasons, you'll have to live with the code duplication or apply one of the hacks.

Someday, asynchronous operations will only be represented with asynchronous APIs. Until then, you have to live in the real world. I hope these techniques will help you adopt async into your existing applications in a way that works best for you. ■

Stephen Cleary *is a husband, father and programmer living in northern Michigan. He has worked with multithreading and asynchronous programming for 16 years and has used async support in the Microsoft .NET Framework since the first CTP. Follow his projects and blog posts at stephencleary.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *James McCaffery and Stephen Toub*

We know how application development can be.

INTRODUCING

Atalasoft MobileImage®

MOBILE CAPTURE SDK

Try it free for 30 days with full support

ATALASOFT.COM

Let DotImage take some of the bite out of your challenges.

Atalasoft DotImage®

Connecting the dots is a no-brainer. DotImage image-enables your .NET-based web application faster, more cost effectively, and less painfully than if done on your own. This proven SDK is versatile, with options including OCR capabilities, WingScan compatibility, and support for a range of formats. Coupled with dedicated assistance from our highly knowledgeable and skilled engineers, DotImage helps your business connect with powerful information hidden inside your documents, making the big picture much easier to see.

Atalasoft A Kofax Company

www.atalasoft.com

Image-enabling experts & bacon connoisseurs. Visit us online to see our full line of SDKs for .NET, Java, and Mobile.

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/redmond

Redmond AUGUST 10 - 14

MICROSOFT HEADQUARTERS, REDMOND, WA

CODE HOME

No code trip would be complete without a stop where it all began, so we're heading to the idyllic **Microsoft Headquarters** in Redmond, WA, **August 10 – 14, 2015!**

Join us as we explore hot topics like Visual Studio, JavaScript/HTML5, ASP.NET, Database and Analytics, and more in over 70+ sessions and workshops. Rub elbows with Microsoft insiders, have lunch with Blue Badges, visit the company store, and experience code at the source.

EVENT SPONSOR Microsoft

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

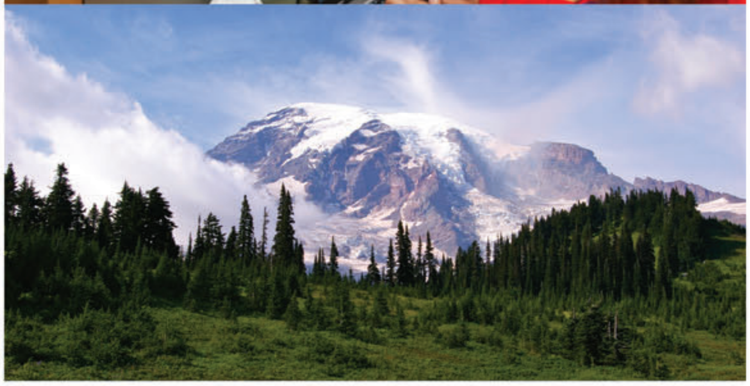
PRODUCED BY 1105 MEDIA INC

vslive.com/redmond

NAVIGATE THE .NET HIGHWAY

DEVELOPMENT TRACKS INCLUDE:

- Visual Studio/.NET
- Web Development
- Design
- Mobile Client
- Windows Client
- Database and Analytics
- Cloud Computing
- Microsoft Sessions



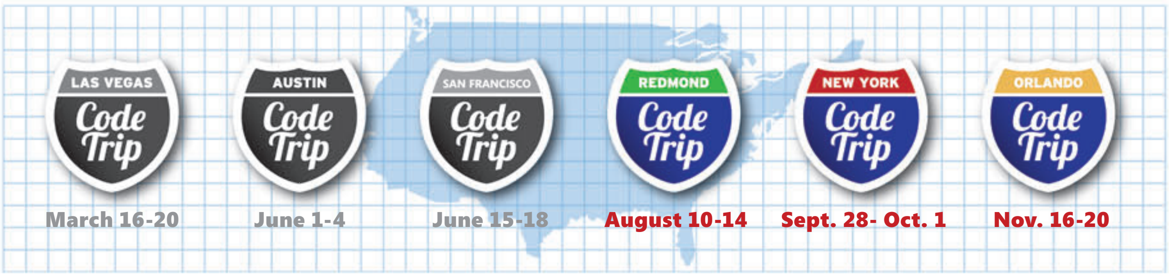
REGISTER BY JULY 8 AND SAVE $300!

Scan the QR code to register or for more event details.

Use promo code REDJUL4

vslive.com/redmond

Join us on the Ultimate Code Trip in 2015!



TURN THE PAGE FOR MORE EVENT DETAILS.

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/redmond

Redmond AUGUST 10 - 14

MICROSOFT HEADQUARTERS, REDMOND, WA

Visual Studio Live! has partnered with the Hyatt Regency Bellevue for conference attendees at a special reduced rate.

REGISTER BY JULY 8 AND SAVE $300!

Scan the QR code to register or for more event details.

Use promo code REDJUL4

vslive.com/redmond

AGENDA AT-A-GLANCE

Cloud Computing | Database and Analytics

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, August 10, 2015	
8:00 AM	12:00 PM	M01 – Workshop: Building Universal Apps for Desktop, Store, and Phone – *Philip Japikse*	
12:00 PM	2:30 PM	Lunch @ The Mixer – Visit the Microsoft Company Store & Visitor Center	
2:30 PM	6:00 PM	M01 – Workshop Continues	
7:00 PM	9:00 PM	Dine-A-Round Dinner	
START TIME	**END TIME**	**Visual Studio Live! Day 1: Tuesday, August 11, 2015**	
8:30 AM	9:30 AM	Keynote: Details Coming Soon	
9:45 AM	11:00 AM	T01 – Azure 10-10: Top 10 Azure Announcements in "T-10" Months – *Vishwas Lele*	T02 – Hack Proofing Your Web Applications – *Adam Tuliper*
11:15 AM	12:30 PM	T06 – Cloud or Not, 10 Reasons Why You Must Know "Web Sites"– *Vishwas Lele*	T07 – Take a Gulp, Make a Grunt, and Call Me Bower – *Adam Tuliper*
12:30 PM	2:30 PM	Lunch – Visit Exhibitors	
2:30 PM	3:45 PM	T11 – Building Mobile Cross-Platform Apps with C# and Xamarin – *Nick Landry*	T12 – AngularJS 101 – *Deborah Kurata*
3:45 PM	4:15 PM	Sponsored Break – Visit Exhibitors	
4:15 PM	5:30 PM	T16 – Building Mobile Cross-Platform Apps in C# with Azure Mobile Services – *Nick Landry*	T17 – AngularJS Forms and Validation – *Deborah Kurata*
5:30 PM	7:00 PM	Microsoft Ask the Experts & Exhibitor Reception	
START TIME	**END TIME**	**Visual Studio Live! Day 2: Wednesday, August 12, 2015**	
8:00 AM	9:00 AM	Keynote: Details Coming Soon	
9:15 AM	10:30 AM	W01 – iOS Develpment - What They Don't Tell You – *Jon Flanders*	W02 – Implementing M-V-VM (Model-View-View Model) for WPF – *Philip Japikse*
10:45 AM	12:00 PM	W06 – Building Cross Platform UI with Xamarin.Forms – *Walt Ritscher*	W07 – To Be Announced
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch – Visit Exhibitors	
1:30 PM	2:45 PM	W11 – Swift for .NET Developers – *Jon Flanders*	W12 – Strike Up a Conversation with Cortana on Windows Phone – *Walt Ritscher*
2:45 PM	3:15 PM	Sponsored Break – Exhibitor Raffle @ 2:55 pm (Must be present to win)	
3:15 PM	4:30 PM	W16 – Creating Applications Using Android Studio – *Kevin Ford*	W17 – Securing Angular Apps – *Brian Noyes*
8:00 PM	10:00 PM	Lucky Strike Evening Out Party	
START TIME	**END TIME**	**Visual Studio Live! Day 3: Thursday, August 13, 2015**	
8:00 AM	9:15 AM	TH01 – Using Multi-Device Hybrid Apps to Create Cordova Applications – *Kevin Ford*	TH02 – Build Data-Centric HTML5 Single Page Applications with Breeze – *Brian Noyes*
9:30 AM	10:45 AM	TH06 – Everything You Always Wanted To Know About REST (But Were Afraid To Ask) – *Jon Flanders*	TH07 – Build Real-Time Websites and Apps with SignalR – *Rachel Appel*
11:00 AM	12:15 PM	TH11 – To Be Announced	TH12 – Knocking it Out of the Park with Knockout.JS – *Miguel Castro*
12:15 PM	2:15 PM	Lunch @ The Mixer – Visit the Microsoft Company Store & Visitor Center	
2:15 PM	3:30 PM	TH16 – Extending XAML to Overcome Pretty Much any Limitation – *Miguel Castro*	TH17 – ASP.NET MVC: All Your Tests Are Belong To Us – *Rachel Appel*
3:45 PM	5:00 PM	TH21 – Agile Database Development – *Richard Hundhausen*	TH22 – Busy JavaScript Developer's Guide to ECMAScript 6 – *Ted Neward*
START TIME	**END TIME**	**Visual Studio Live! Post-Conference Workshops: Friday, August 14, 2015**	
8:00 AM	12:00 PM	F01 – Workshop: SQL Server 2014 for Developers – *Andrew Brust and Leonard Lobel*	
12:00 PM	1:00 PM	Lunch	
1:00 PM	5:00 PM	F01 – Workshop Continues	

Sessions and speakers subject to change.

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

DETAILS COMING SOON!

With all of the announcements coming out of Build and Ignite, we'll be finalizing the Redmond Keynotes and the **FULL Track** of Microsoft-led sessions shortly. Check **vslive.com/redmond** for updates!

UNDER CONSTRUCTION

NAVIGATE THE .NET HIGHWAY

Design	Mobile Client	Visual Studio / .NET	Web Development	Windows Client	Microsoft Sessions

(Separate entry fee required)		
M02 – Workshop: UX Design Process Essentials– Steps and Techniques – *Billy Hollis*	**M03 – Workshop: ALM and DevOps with the Microsoft Stack –** *Brian Randell*	
M02 – Workshop Continues	**M03 – Workshop Continues**	
T03 – UX Design Principle Fundamentals for Non-Designers – *Billy Hollis*	**T04 – A Lap Around Visual Studio 2015 –** *Robert Green*	**T05 – Microsoft Session, Details Coming Soon**
T08 – Designing and Building UX for Finding and Visualizing Data in XAML Applications – *Billy Hollis*	**T09 – What's New in ALM –** *Brian Randell*	**T10 – Microsoft Session, Details Coming Soon**
T13 – Windows, NUI and You – *Brian Randell*	**T14 – Hosting ASP.NET Applications Cross Platform with Docker –** *Adam Tuliper*	**T15 - Microsoft Session, Details Coming Soon**
T18 – Building Windows 10 LOB Apps – *Robert Green*	**T19 – SOLID Design Patterns for Mere Mortals –** *Philip Japikse*	**T20 – Microsoft Session, Details Coming Soon**
W03 – Moving Web Apps to the Cloud – *Eric D. Boyd*	**W04 – Enhancing Application Quality Using Visual Studio 2015 Premium Features –** *Anthony Borton*	**W05 – Microsoft Session, Details Coming Soon**
W08 – Solving Security and Compliance Challenges with Hybrid Clouds – *Eric D. Boyd*	**W09 – Load Testing ASP.NET & WebAPI with Visual Studio –** *Benjamin Day*	**W10 – Microsoft Session, Details Coming Soon**
W13 – To Be Announced	**W14 – To Git or Not to Git for Enterprise Development –** *Benjamin Day*	**W15 – Microsoft Session, Details Coming Soon**
W18 – Building for the Internet of Things: Hardware, Sensors & the Cloud – *Nick Landry*	**W19 – Not Your Grandfather's Build – A Look at How Build Has Changed in 2015 –** *Anthony Borton*	**W20 – Microsoft Session, Details Coming Soon**
TH03 – Database Development with SQL Server Data Tools – *Leonard Lobel*	**TH04 – Stop the Waste and Get Out of (Technical) Debt –** *Richard Hundhausen*	**TH05 – Microsoft Session, Details Coming Soon**
TH08 – Programming the T-SQL Enhancements in SQL Server 2012 – *Leonard Lobel*	**TH09 – What's New in C# 6.0 –** *Jason Bock*	**TH10 – Microsoft Session, Details Coming Soon**
TH13 – Power BI 2.0: Analytics in the Cloud and in Excel – *Andrew Brust*	**TH14 – Async and Await Best Practices –** *Mark Rosenberg*	**TH15 – Microsoft Session, Details Coming Soon**
TH18 – Busy Developer's Guide to NoSQL – *Ted Neward*	**TH19 – Microsoft's .NET is Now Open Source and Cross-Platform. Why it Matters. –** *Mark Rosenberg*	**TH20 – Microsoft Session, Details Coming Soon**
TH23 – Big Data and Hadoop with Azure HDInsight – *Andrew Brust*	**TH24 – Managing the .NET Compiler –** *Jason Bock*	**TH25 – Microsoft Session, Details Coming Soon**
(Separate entry fee required)		
F02 – Workshop: Service Oriented Technologies: Designing, Developing, & Implementing WCF and the Web API – *Miguel Castro*		
F02 – Workshop Continues		

vslive.com/redmond

Analyze User Behavior in Your Windows/Windows Phone App with Google Analytics

Nicola Delfino

App development is by definition an iterative process, so you'll want to understand, in the shortest possible time, how your app is used and how a new version or feature may impact user behavior. How much is a feature really used? What is the user behavior on a specific page? How much time does a user spend to accomplish a specific task? What are the most common hardware configurations? What's going on when a crash happens? How successful are trial versions and in-app purchases? Do your users run the app offline? These are only some of the questions a telemetry system can answer.

There are a number of telemetry providers that officially support Windows development (bit.ly/1KGgAdQ), and Microsoft Azure includes Visual Studio Application Insight, which is also integrated in Visual Studio 2015 (bit.ly/1EeM8Ui).

This article discusses:

- Integrating a Windows/Windows Phone app with Google Analytics
- Tracking user navigation and app version
- Optimizing an app using A/B testing

Technologies discussed:

Microsoft Azure, Visual Studio 2015, Windows, Windows Phone, Google Analytics

Code download available at:

msdn.microsoft.com/magazine/msdnmag0715

Google Analytics, thanks to the Google Analytics SDK for Windows (available on Codeplex at bit.ly/1zXfvxJ), is also easy to integrate into a Windows or Windows Phone app (both Silverlight and Universal).

In this article, I'll focus on Google Analytics, because it's one of the most widely used providers on other platforms, including the Web, and it's a great choice if you've already instrumented a Web site with Google and want to use the same account for apps. I'll describe how to collect telemetry information with Google Analytics, and analyze the collected data, such as installed app versions, geographic distribution, and user behaviors, and how to perform an A/B Testing.

Understanding telemetry concepts is useful no matter which provider you use. A great starting point on this topic is Kraig Brockschmidt's deck from Build 2014 (bit.ly/1QGIsS0).

What Is Google Analytics?

Google Analytics was originally built to track and analyze user behaviors on Web sites. Over time, Google extended its support and tools to apps, and now an SDK is available for iOS and Android, as well. Google also provides a measurement protocol that lets you make HTTP calls to send raw user interaction data directly to Google Analytics servers (bit.ly/1kHgDYz). This allows you to determine how users interact with their businesses from almost any environment. The Google Analytics SDK uses this protocol for Windows 8 and Windows Phone. The SDK is an independent, open source tool set that aims to have feature parity and API similarity with the official Google Analytics SDK for Android.

Integrate Your App with Google Analytics

To integrate an app, first you need to access the Google Analytics dashboard with a valid Google account, then create at least one application account that can access the administrative area (bit.ly/1kPLnqG). It's also a good idea to create separate application accounts for development, beta testing and production. This way, you keep the data from each set separate, and avoid polluting your important production data with bad data from other developers and beta testers.

An application account uniquely identifies an application (Contoso, for the purposes of this article). In this context, an "application" really refers to a collection of apps available for different platforms and/or Web sites (Contoso for Windows, Contoso for Android, Contoso Web dashboard), all of which send their telemetry information to the same account. Each application may also be available in multiple versions (Contoso for Windows 1.0, Contoso for Windows 1.1, Contoso for Android 2.0.1).

When you create the account in Google Analytics, you also need to define one or more properties. Here, a property is an app or a Web site for your account. An app property can have multiple versions, and it's logical to have a property for each supported platform (for example, ContosoW8, ContosoWP8, ContosoDroid). After you've chosen names for your app and answered a few questions to classify the business scope of the app, Google Analytics generates a numeric account ID and a tracking ID with the format UA-xxxxxxxx-x, which represents your platform-specific app in Google Analytics and will be used in all telemetry communications sent from the app.

In my sample scenario, I have a universal application (W8.1 and WP8.1), so I created two property IDs in one Google Analytics account. To enable my app to talk with Google Analytics, I'll use a NuGet package. To do this, for both the Windows and Windows Phone apps, right-click on References, select manage NuGet packages, and add the package "GoogleAnalyticsSDK." This package adds a couple of references and the file analytics.xml, shown in **Figure 1**, to your Visual Studio solution. You'll have to edit this file so the app can use the correct tracking ID you created earlier.

Finally, you must ensure your app is authorized to use the network. For the Windows Phone app, select the ID_CAP_NETWORKING capability in the WMAppManifest.xml file; for Windows 8, select the Internet (client) capability in the package.appxmanifest file.

Instrumenting the App

The first category of information most developers want to track is user navigation. Using this SDK, you need to send the correct event every time a user accesses a page. The Google Analytics SDK exposes the SendView command for this, and the natural place to call this method is the Loaded event of each page:

```
void MainPage_Loaded(object sender, RoutedEventArgs e)
{
GoogleAnalytics.EasyTracker.GetTracker().SendView("Main");
}
void Page1_Loaded(object sender, RoutedEventArgs e)
{
GoogleAnalytics.EasyTracker.GetTracker().SendView("Page1");
}
void Page2_Loaded(object sender, RoutedEventArgs e)
{
GoogleAnalytics.EasyTracker.GetTracker().SendView("Page2");
}
```

Keep in mind that a "view" is not just an app page; you also need to consider other aspects of the UI that focus the user's attention (such as the settings panel and dialog boxes) that you want to monitor and analyze.

SendView requires a string parameter that uniquely identifies the page or UI aspect you're tracking. This name will appear on the Google Analytics dashboard and on reports as the reference for your page. It should be generic and meaningful and easily associable with the source page. It must not be translated even if your app supports multiple languages because it would be interpreted by the analysis system as a different page. In other words, unless you specifically want to differentiate telemetry data for localized versions of your app, make sure the identifiers you send to Google Analytics are not themselves localized.

Now the app is ready to send its first page-navigation telemetry information. To see it, open your app and navigate through the pages for a while.

User Navigation and the App Version

Within a few seconds, the navigation data is ready and available for your analysis. Go to google.com/analytics and sign in to access your homepage. Then click on the All mobile App Data link below the property you want to analyze (in my sample, the property is ContosoW8 or Contoso WP8).

The welcome dashboard shows historical data for the last 30 days, not including the current day. If the account is new, most likely all charts will be empty. There's also a dashboard area dedicated to the most recent data, which you can access by selecting the Real Time button on the left side of the dashboard.

The Overview page indicates the number of active users, and shows the screen view in real time with a detailed graph for the last minute and the last hour (see **Figure 2**). Vertical bars on these graphs show user page accesses.

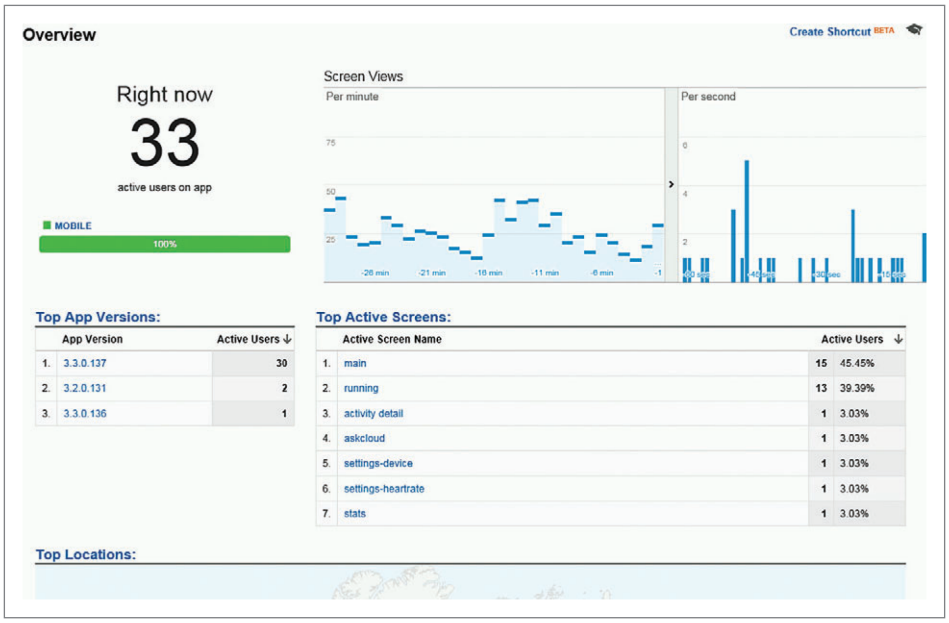
You can also see a list of the most active screens in the last hour, with the number of active users. The name you'll see on the dashboard is the one used as the SendView parameter in the app code. **Figure 2** shows real-time sample data in "Caledos Runner," an app I developed that uses Google Analytics to collect usage data. I wanted to show real data in the app so you could see how much information you can get from this tool—and the corresponding potential.

Each screen name is clickable so you can slice the data and analyze the specific use of a page. One of the most important pieces of information here is the time spent on each page (or, better, on each view). After clicking a screen name, you can add a metric

Figure 1 **Analytics.xml for Windows and Windows Phone App**

```
<?xml version="1.0" encoding="utf-8" ?>
<analytics xmlns="http://googleanalyticssdk.codeplex.com/ns/easytracker">
<trackingId>UA-60759067-2</trackingId>
  <appName>ContosoWindows</appName>
  <appVersion>1.0.0.0</appVersion>
</analytics>

<?xml version="1.0" encoding="utf-8" ?>
<analytics xmlns="http://googleanalyticssdk.codeplex.com/ns/easytracker">
  <trackingId>UA-60759067-1</trackingId>
  <appName>ContosoWinPhone</appName>
  <appVersion>1.0.0.0</appVersion>
</analytics>
```



Top App Versions:

	App Version	Active Users ↓
1.	3.3.0.137	30
2.	3.2.0.131	2
3.	3.3.0.136	1

Top Active Screens:

	Active Screen Name	Active Users ↓	
1.	main	15	45.45%
2.	running	13	39.39%
3.	activity detail	1	3.03%
4.	askcloud	1	3.03%
5.	settings-device	1	3.03%
6.	settings-heartrate	1	3.03%
7.	stats	1	3.03%

Figure 2 **Real-Time Data Dashboard**

showing average time on screen to the graph, by clicking Select a Metric from the Explorer tab.

One great tool Google provides is the Behavior Flow report, shown in **Figure 3**. Thanks to the SendView API, this tool is available by selecting Behavior from the Reporting tab on the dashboard, and it lets you analyze how users move across your app's pages from the beginning of a user session. The graph is composed of a sequence of screens (first screen, second screen and so on), and shows the number of user sessions that connect and exit. For the session exit, you can see the next page accessed (ordered by session number) and the number of users who leave the app from that page (in red).

This tool allows you to understand where users spend their time when they use the app, and you may eventually decide to make access to popular pages easier by providing a bigger, more recognizable button. **Figure 3** shows that in my app, the top three next screen destinations from the main page are askcloud, running and activity detail. This suggests it's probably a good idea to give a central and visible position to the corresponding buttons on the main page, or to move some of the functionality users find on those pages to the main screen in order to reduce the time and the number of taps users need to accomplish these activities.

You'll also want to know which version of your app is being used, and this information comes from the devices using your app. On each call, the SDK sends the app version provided in analytics.xml so you can slice your data and find the version used. Historical app version data is available under Audience | App Versions in the left pane of the dashboard. **Figure 4** shows what typically happens when I release a new version of my app to the store. Because the default behavior on Windows Phone 8.1 is to automatically download and install the new version of the app when available, within a few days more than 90 percent of the user base migrates to the latest version, (though a long tail of less than 10 percent of users remains who, for some reason, can't or don't want to upgrade).

Other dimensions you receive "for free" with navigation tracking are the location, language, device name (which isn't 100 percent accurate for Windows Phone) and network service provider, while on the measures side you can find the number of sessions, unique users and unique new users.

With this information, you're able to answer questions such as: What's the most active country or city? Where have I had the most new users in the last month? With this new version, are my customers using the app more or less? And in this context, the only limit is your imagination because you can set custom events, counters, dimensions and metrics.

How a user navigates from one screen to another is a good starting point for understanding how your app is used, but there's much more information available, more related to the specific domain of your app. This type of information can be tracked as a custom event, which usually involves some generic user interaction or information that makes sense to be bubbled up to the developer

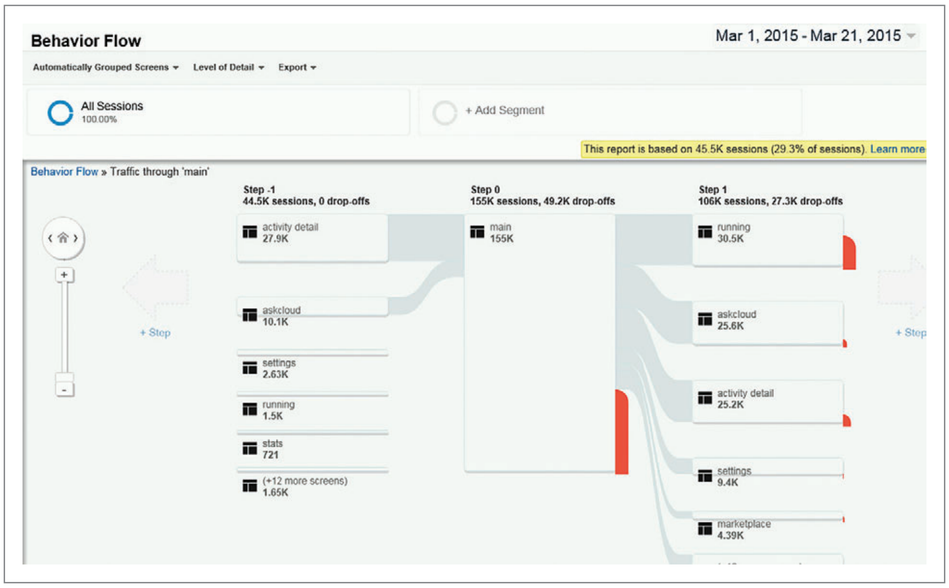


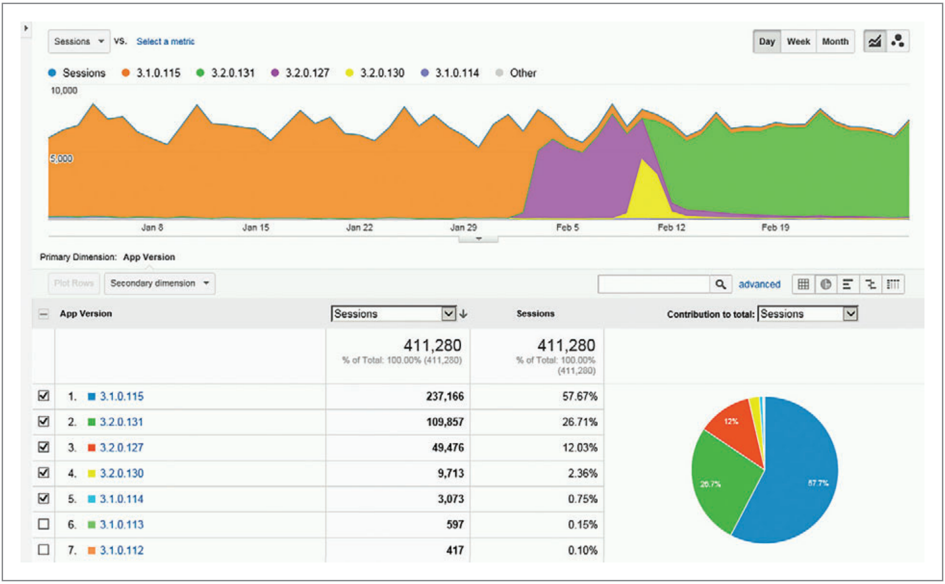
Figure 3 **A Sample Behavior Flow Report**

team. An event consists of four fields you can use to describe a user's interaction with your app content:

- String Category
- String Action
- String Label
- Long Value

Because all custom events go in a single common list, the category, action and label fields help to identify specific information you want to analyze.

The information collected in this way is very generic, and this is where telemetry becomes an art. To give meaning to the data, you start with the question you want to answer, which leads to the telemetry design, which then leads to the instrumentation you need. So let's start with an example approach.



Primary Dimension: App Version

	App Version	Sessions	Sessions	Contribution to total: Sessions
		411,280 % of Total: 100.00% (411,280)	411,280 % of Total: 100.00% (411,280)	
☑	1. 🟦 3.1.0.115	237,166	57.67%	
☑	2. 🟩 3.2.0.131	109,857	26.71%	
☑	3. 🟥 3.2.0.127	49,476	12.03%	
☑	4. 🟨 3.2.0.130	9,713	2.36%	
☑	5. 🟦 3.1.0.114	3,073	0.75%	
☐	6. 🟩 3.1.0.113	597	0.15%	
☐	7. 🟧 3.1.0.112	417	0.10%	

Figure 4 **App Version Data**

Question: My sample app has many functions, triggered by buttons on various pages of the app. I want to understand which functions are used more, to focus my development effort in the right direction.

Telemetry Design: The main functions of the app are triggered by buttons, so if I track the clicks on those buttons, I'll probably gain a better understanding of user preferences. The telemetry's success metric will be the number of clicks I receive from each button.

Instrumentation: I can go to the click event of the selected buttons and track the event.

The following code shows how to record that a user has clicked the "play" and "stop" buttons:

```
private void buttonPlay_Click(object sender, RoutedEventArgs e)
{
GoogleAnalytics.EasyTracker.GetTracker().SendEvent(
  "ui_action", "press", "play", 0);
// Your code here
}

private void buttonStop_Click(object sender, RoutedEventArgs e)
{
GoogleAnalytics.EasyTracker.GetTracker().SendEvent(
  "ui_action", "press", "stop", 0);
// Your code here
}
```

As in the previous user navigation analysis, you have an Events table under both the Real-Time and Behavior groups. Depending on the type of analysis you want to do, you can select the appropriate view, historical or real time.

Events are shown for a specific time range (the default is one month), so you can see the number of events grouped by the top category, action and label in it. You can drill down on a specific category and on a specific event action and you can also change the date interval for the analysis.

For a specific event category, action or label, you can find the total number of events, the number of unique events, the sum of all the values and the corresponding average value.

Figure 5 shows a sample list of events, and **Figure 6** shows the total, unique, sum and average values for the group Grp1.

A real-world example of use can help you better understand the potential of the tool. For "Caledos Runner," I wanted to understand how much the app was really used for tracking activities, and because I didn't want to force the user to register for an account in order to use the application, I used events to track this information.

On app start, I tracked an event for each registered user, and a different event for each user who isn't registered, with code similar to the following:

```
if (!string.IsNullOrEmpty(currentUser.AccessToken))
{
GoogleAnalytics.EasyTracker.GetTracker().SendEvent(
  "user", "Cloud", "Registered", 1);
}
else
{
GoogleAnalytics.EasyTracker.GetTracker().SendEvent(
  "user", "Cloud", "Unregistered", 1);
}
```

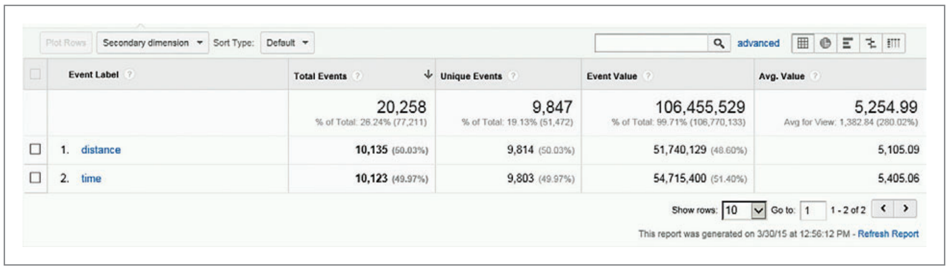
I've discovered that 55 percent of users don't send their data to the cloud, so in order to get a precise profile of my users, at the end of each GPS activity the app sends one event with information about the distance covered, and one event with the total elapsed time of the activity, using code similar to the following:

Figure 5 **Sample Events**

Group	Action	Label	Value
Grp1	Press	Button1	1
Grp1	Press	Button1	2
Grp1	Press	Button1	2
Grp1	Press	Button2	3
Grp1	Tap	Label1	4

Figure 6 **Summary Values for Grp1**

Total Events	5
Unique Events	4
Event value	12
Event average value	2.4



	Event Label	Total Events	Unique Events	Event Value	Avg. Value
		20,258 % of Total: 26.24% (77,211)	9,847 % of Total: 19.13% (51,472)	106,455,529 % of Total: 99.71% (106,770,133)	5,254.99 Avg for View: 1,382.84 (280.02%)
☐	1. distance	10,135 (50.03%)	9,814 (50.03%)	51,740,129 (48.60%)	5,105.09
☐	2. time	10,123 (49.97%)	9,803 (49.97%)	54,715,400 (51.40%)	5,405.06

Show rows: 10 Go to: 1 1 - 2 of 2

This report was generated on 3/30/15 at 12:56:12 PM - Refresh Report

Figure 7 **Custom Events Sample**

```
GoogleAnalytics.EasyTracker.GetTracker().SendEvent(
  "activity", "outdoor", "time", (long)CurrentFitnessActivity.Duration);
GoogleAnalytics.EasyTracker.GetTracker().SendEvent(
  "activity", outdoor", "distance", (long)model.CurrentFitnessActivity.
TotalDistance);
```

As a result, I've obtained the data shown in **Figure 7** for the first week of February 2015.

What this analysis says is: In the first week of February, the app has successfully tracked more than 15,000 hours (54,000,000 seconds) and 51,000 Km (32,000 miles). The average user uses the app for an activity of about 5 Km (3.1 miles), covering the distance in an hour and a half (5,400 seconds).

Good to know, but not enough! This is just an average usage view, but more likely you'd want to know how that usage is really distributed. This is a multisport app, so you'd want to know the most popular sport and whether that popularity comes from a small number of users using the app often or a lot of users using it, but for only a short amount of time.

This is where Google Analytics custom dimensions and metrics can help. Custom dimensions and metrics allow you to bring additional data into Google Analytics that can help you answer new questions about how users are interacting with your content. A metric is a count of a data type that's summed up across your Google Analytics hits. A metric corresponds to a column in a report. Dimensions allow you to break down a metric by a particular value, for example, screen views by screen name. Dimensions correspond to the rows in a report (bit.ly/1I0ry2r).

There are two main steps to implement a custom dimension or metric:

- Define the custom dimension or metric using the Google Analytics Web interface (bit.ly/1Je9RJQ).
- Implement code to set and collect custom dimension and metric values.

Each Google Analytics property has 20 available indices for custom dimensions, and another 20 indices for custom metrics. Once a dimension or a metric is created, your code can refer to that metric or dimension using the corresponding index.

Getting back to my question, to understand the sport breakdown I created a Dimension, Activity Type, with scope, Hit, and used the following code to track this information on the view shown by the app during a fitness session:

```
GoogleAnalytics.EasyTracker.GetTracker().SetCustomDimension(1, activityType);
GoogleAnalytics.EasyTracker.GetTracker().SendView("Running");
```

A/B Testing

A/B testing is a term marketers use to refer to testing that provides insight into visitor behavior, with the aim of increasing the conversion rate. (According to Wikipedia, the conversion rate "is the proportion of visitors of a Web site who take action to go beyond a casual content view or Web site visit, as a result of subtle or direct requests from marketers, advertisers, and content creators.")

In the app development world, A/B testing means testing two different versions of an element (such as a page or a button) using a metric that defines the success of one scenario over another.

Suppose you have two designs of a page, A and B. Typically, A is the existing design (the control), and B is the new design. You can split your user base between these two versions and measure their performance using a metric that makes sense for your app (perhaps clicking on a specific button) to find your success metric or your conversion rate. At the end of the test, you select the element that performs best.

In other words, suppose each design is a different app page with its XAML file (page3v1.xaml and page3v2.xaml). On each page you have your target—the button you want the user to click—and you can use the click event of the button to track a custom event that recognizes a successful click of the button. By taking advantage of the MVVM pattern, you can decouple the view from model and the view model, so you can easily have two XAML files that contain the same logic and data.

You can then use code like the following to bring users to the pages to test, splitting navigation evenly between versions A and B of the page:

```
if (DateTime.Now.Second % 2 == 0)
  this.Frame.Navigate(typeof(Page3V1));
else
  this.Frame.Navigate(typeof(Page3V2));
```

You can track the success of the layout by recording the button click using:

```
GoogleAnalytics.EasyTracker.GetTracker().SendEvent(
  "ABTest", "Scenario1", "pageV1", 0);
```

or the button click on the first page:

```
GoogleAnalytics.EasyTracker.GetTracker().SendEvent(
  "ABTest", "Scenario1", "pageV2", 0);
```

and on the click of the same button on the second page.

The total number of events for each label (PageV1, PageV2) determines the most successful page. At the end of the test, only that page will remain in the application, while the other can be removed.

Wrapping up

In modern app development, telemetry and analysis tools can make a real difference in terms of how quickly you can identify user behaviors, as well as application problems and their resolution. In this article, I've discussed only a small sample of the scenarios that can be analyzed. The number and type of such scenarios is limited only by your imagination and curiosity. ■

Nicola Delfino *is an application development manager in the Microsoft Italian services division and the author of "Caledos Runner," one of most successful fitness-tracking apps for Windows Phone, which gave him the opportunity to use telemetry to discover and analyze how people use his app. You can reach him at nicold@microsoft.com or nicola@caledos.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Kraig Brockschmidt*

Spreadsheets Made Easy.

Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

WIN Windows Forms | Silverlight | WPF WPF

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.

Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.

Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

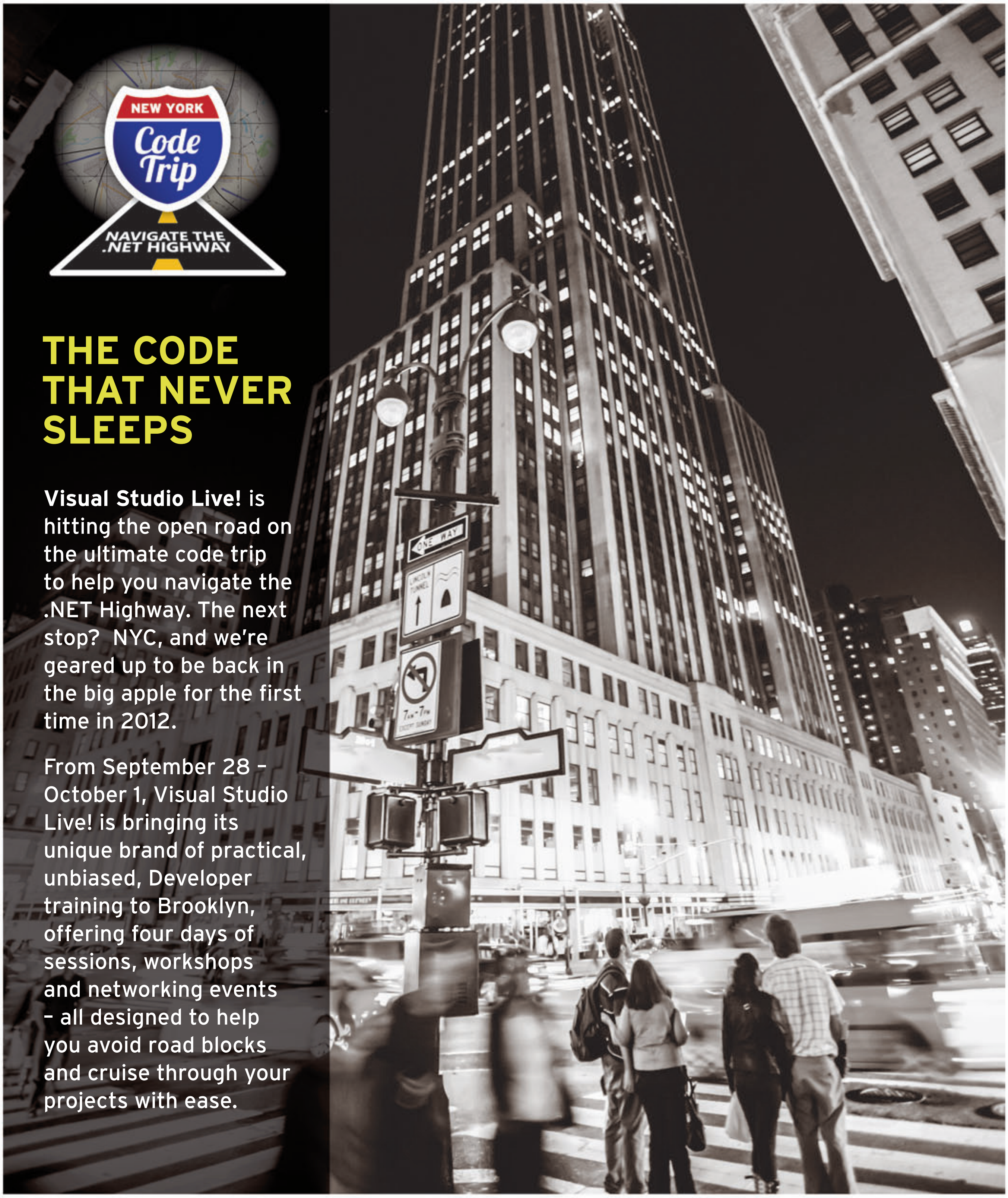
Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

VSLIVE.COM/NEWYORK

New York

SEPTEMBER 28 – OCTOBER 1

MARRIOTT @ BROOKLYN BRIDGE • NEW YORK, NY



THE CODE THAT NEVER SLEEPS

Visual Studio Live! is hitting the open road on the ultimate code trip to help you navigate the .NET Highway. The next stop? NYC, and we're geared up to be back in the big apple for the first time in 2012.

From September 28 – October 1, Visual Studio Live! is bringing its unique brand of practical, unbiased, Developer training to Brooklyn, offering four days of sessions, workshops and networking events – all designed to help you avoid road blocks and cruise through your projects with ease.

GOLD SPONSOR GitHub

SUPPORTED BY Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA INC

NAVIGATE THE .NET HIGHWAY

DEVELOPMENT TOPICS INCLUDE:

- VISUAL STUDIO / .NET
- WEB DEVELOPMENT
- DESIGN
- MOBILE CLIENT
- WINDOWS CLIENT
- DATABASE AND ANALYTICS
- CLOUD COMPUTING

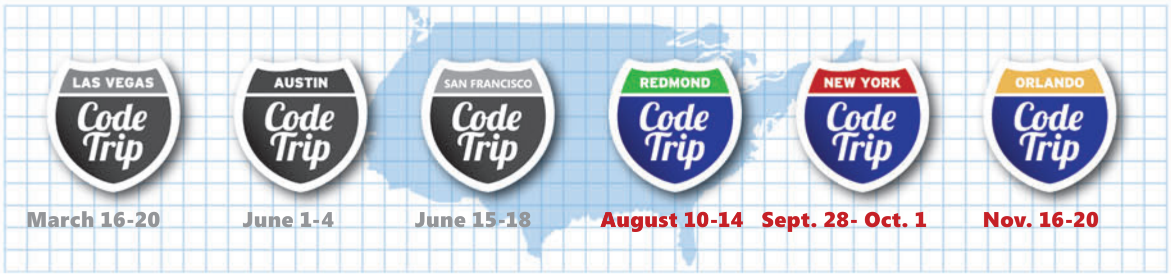
Register by August 5 and Save $300!

USE PROMO CODE NYJUL4

SCAN THE QR CODE TO REGISTER OR FOR MORE EVENT DETAILS.

VSLIVE.COM/NEWYORK

Join us on the Ultimate Code Trip in 2015!



TURN THE PAGE FOR MORE EVENT DETAILS.

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

VSLIVE.COM/NEWYORK

New York

SEPTEMBER 28 – OCTOBER 1

MARRIOTT @ BROOKLYN BRIDGE • NEW YORK, NY

THE HOST HOTEL

The New York Marriott at the Brooklyn Bridge

333 Adams Street
Brooklyn, NY 11201

Special attendee rate: $289*

Book by: September 9, 2015

Located in Renaissance Plaza, the New York Marriott at the Brooklyn Bridge is one block away from nine major subway lines. Take the subway to Manhattan in just 5 minutes, or wander through Brooklyn's best-kept secrets—the Brooklyn Heights Promenade, the sculpture garden at Brooklyn's famous Pratt Institute, and the ultra-hip DUMBO and Williamsburg neighborhoods.

* Rooms are subject to availability and applicable state, local and occupancy taxes.

FEATURED KEYNOTE SPEAKERS

Brian Harry, Corporate Vice President, Microsoft

Mary Jo Foley, Journalist and Author

Register by August 5 and Save $300!

USE PROMO CODE VSLJUL4

SCAN THE QR CODE TO REGISTER OR FOR MORE EVENT DETAILS.

VSLIVE.COM/NEWYORK

FOLLOW US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

New York AGENDA AT-A-GLANCE

Cloud Computing	Database and Analytics	Design	Mobile Client	Visual Studio / .NET	Web Development	Windows Client

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, September 28, 2015 (Separate entry fee required)		
9:00 AM	6:00 PM	**M01 - Workshop: Big Data, Analytics and NoSQL: Everything You Wanted to Learn But Were Afraid to Ask** - *Andrew Brust*	**M02 - Workshop: Native Mobile App Development for iOS, Android and Windows Using C#** - *Marcel de Vries and Roy Cornelissen*	**M03 - Workshop: ALM and DevOps with the Microsoft Stack** - *Brian Randell*
6:45 PM	9:00 PM	**Dine-A-Round**		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, September 29, 2015			
8:00 AM	9:00 AM	**KEYNOTE: To Be Announced** - *Brian Harry, Corporate Vice President, Microsoft*			
9:15 AM	10:30 AM	**T01 - AngularJS 101** *- Deborah Kurata*	**T02 - Azure 10-10: Top 10 Azure Announcements in "T-10" Months** *- Vishwas Lele*	**T03 - UX Design Principle Fundamentals for Non-Designers** *- Billy Hollis*	**T04 - From ASP.NET Site to Mobile App in About an Hour** *- Ryan J. Salva*
10:45 AM	12:00 PM	**T05 - AngularJS Forms and Validation** - *Deborah Kurata*	**T06 - Cloud or Not, 10 Reasons Why You Must Know "Web Sites"** *- Vishwas Lele*	**T07 - Designing and Building UX for Finding and Visualizing Data in XAML Applications** *- Billy Hollis*	**T08 - Microsoft Session To Be Announced**
12:00 PM	1:30 PM	**Lunch — Visit Exhibitors**			
1:30 PM	2:45 PM	**T09 - Building Mobile Cross-Platform Apps with C# and Xamarin** - *Nick Landry*	**T10 - Inside the Azure Resource Manager** - *Michael Collier*	**T11 - Windows, NUI and You** *- Brian Randell*	**T12 - Microsoft Session To Be Announced**
3:00 PM	4:15 PM	**T13 - Building Mobile Cross-Platform Apps in C# with Azure Mobile Services** - *Nick Landry*	**T14 - Automating Your Azure Environment** - *Michael Collier*	**T15 - Building Windows 10 LOB Apps** - *Billy Hollis*	**T16 - Defensive Coding Techniques in C#** *- Deborah Kurata*
4:15 PM	5:30 PM	**Welcome Reception**			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, September 30, 2015			
8:00 AM	9:00 AM	**KEYNOTE: Microsoft 3.0: New Strategy, New Relevance** *- Mary Jo Foley, Journalist and Author; with Andrew Brust, Senior Director, Datameer*			
9:15 AM	10:30 AM	**W01 - iOS Development - What They Don't Tell You** *- Jon Flanders*	**W02 - Moving Web Apps to the Cloud** - *Eric D. Boyd*	**W03 - XAML Antipatterns** *- Ben Dewey*	**W04 - What's New in C# 6.0** *- Jason Bock*
10:45 AM	12:00 PM	**W05 - Swift for .NET Developers** *- Jon Flanders*	**W06 - Solving Security and Compliance Challenges with Hybrid Clouds** - *Eric D. Boyd*	**W07 - Extending XAML to Overcome Pretty Much Any Limitation** - *Miguel Castro*	**W08 - Using Microsoft Application Insights to Implement a Build, Measure, Learn Loop** - *Marcel de Vries*
12:00 PM	1:30 PM	**Birds-of-a-Feather Lunch — Visit Exhibitors**			
1:30 PM	2:45 PM	**W09 - Building Cross-Platform C# Apps with a Shared UI via Xamarin.Forms** - *Nick Landry*	**W10 - To Be Announced**	**W11 - Real World SQL Server Data Tools** - *Benjamin Day*	**W12 - Enhancing Application Quality Using Visual Studio 2015 Premium Features** - *Anthony Borton*
3:00 PM	4:15 PM	**W13 - Creating Applications Using Android Studio** - *Kevin Ford*	**W14 - ASP.NET MVC: All Your Tests Are Belong To Us** *- Rachel Appel*	**W15 - Transact-SQL for Application Developers - Attendees Choose Topics** *- Kevin Goff*	**W16 - Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI** *- Marcel de Vries*
4:30 PM	5:45 PM	**W17 - Using Multi-Device Hybrid Apps to Create Cordova Applications** - *Kevin Ford*	**W18 - Build Data-Centric HTML5 Single Page Applications with Breeze** - *Brian Noyes*	**W19 - SQL Server Reporting Services - Attendees Choose Topics** - *Kevin Goff*	**W20 - Managing the .NET Compiler** - *Jason Bock*
7:00 PM	9:00 PM	**Visual Studio Live! Evening Event**			

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, October 1, 2015			
8:00 AM	9:15 AM	**TH01 - Everything You Always Wanted To Know About REST (But Were Afraid To Ask)** *- Jon Flanders*	**TH02 - Securing Angular Apps** *- Brian Noyes*	**TH03 - Implementing Data Warehouse Patterns - Attendees Choose** - *Kevin Goff*	**TH04 - Not Your Grandfather's Build - A Look at How Build Has Changed in 2015** *- Anthony Borton*
9:30 AM	10:45 AM	**TH05 - Comparing Performance of Different Mobile Platforms** *- Kevin Ford*	**TH06 - I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(Me), Maybe?** - *Rachel Appel*	**TH07 - Power BI 2.0: Analytics in the Cloud and in Excel** *- Andrew Brust*	**TH08 - Async Patterns for .NET Development** - *Ben Dewey*
11:00 AM	12:15 PM	**TH09 - To Be Announced**	**TH10 - Build Real-Time Websites and Apps with SignalR** *- Rachel Appel*	**TH11 - Busy Developer's Guide to NoSQL** - *Ted Neward*	**TH12 - DevOps and ALM-Better Together Like Peanut Butter and Chocolate** - *Brian Randell*
12:15 PM	1:30 PM	**Lunch**			
1:30 PM	2:45 PM	**TH13 - WCF & Web API: Can We All Just Get Along?!?** *- Miguel Castro*	**TH14 - Busy JavaScript Developer's Guide to ECMAScript 6** - *Ted Neward*	**TH15 - Big Data and Hadoop with Azure HDInsight** - *Andrew Brust*	**TH16 - To Git or Not to Git for Enterprise Development** *- Benjamin Day*
3:00 PM	4:15 PM	**TH17 - Recruiters: The Good, The Bad, & The Ugly** *- Miguel Castro*	**TH18 - Busy Developer's Guide to MEANJS** - *Ted Neward*	**TH19 - Predictive Analytics and Azure Machine Learning** *- Andrew Brust*	**TH20 - Load Testing ASP.NET & WebAPI with Visual Studio** *- Benjamin Day*

Sessions and speakers subject to change.

vslive.com/newyork

Event Hubs for Analytics and Visualization, Part 3

Bruno Terkaly

If you've been following this series, you'll finally see the realization of the main goal of the previous work in this final installment—to visualize data originating from Raspberry Pi devices in the field. This is the final installment of a three-part series around an Internet of Things (IoT) scenario. For the previous two articles, check the April issue (msdn.microsoft.com/magazine/dn948106) and June issue (msdn.microsoft.com/magazine/mt147243).

The high-level goals for this month's column are quite simple. I'll build a Node.js application that acts as a Web site and provides data to the mobile device, which will then display a bar chart of rainfall data. I need the Node.js Web server because it's generally not advisable for a mobile device to directly access data stores.

Other technologies could have performed this function, such as ASP.NET Web API or Microsoft Azure API Apps. Most architects would advise you to filter, sort and parse data in the middle tier on a server, not on the mobile device itself. That's because mobile devices have less computing power and you don't want to send large amounts of information across the wire to the device. In short, let the server do the heavy lifting and let the device display visuals.

Get Started

To run the examples in this month's column, you'll need to have completed the previous two from the April and June issues. If you haven't, you can probably get by with manually building out your own DocumentDB data store.

This article discusses:

- Completing the Internet of Things project
- Creating a Node.js app to display Web data
- Visualizing the data on a mobile client

Technologies discussed:

Microsoft Azure, ASP.NET Web API, Node.js, Raspberry Pi, Internet of Things

Code download available at:

bit.ly/1KBqq3Q

Assuming you've previously created a DocumentDB data store, start by going to the Azure Portal and viewing the DocumentDB database. There's some nice tooling at the portal, which lets you view your data and perform different queries. Once you validate the data is there and understand its basic structure, you can begin to build the Node.js Web server.

Developers generally like to validate software as they build it out. A lot of developers start with unit testing, for example, with tools such as Mocha or Jasmine. For example, before building the mobile client, it makes sense to ensure the Node.js Web server operates as expected. One approach is to use a Web proxy tool, such as Fiddler. That makes it easy to issue Web requests (such as HTTP GET) and view the response in native JSON format. This can be useful, because when building the mobile client, you can be confident any problems are related to the mobile client, not the Web service.

To keep things simple, use a Windows Phone device as the client, even though iOS and Android devices are more prevalent. Perhaps the best approach would be a Web-based technology. This would let you consume or visualize the data from any device, therefore not limiting yourself to mobile devices. Another approach is to use native cross-platform products, such as Xamarin.

There are several sources of information you can use to build out a Web-based mobile client, not the least of which is the W3C standards body. You can read more about them in the standards documents at bit.ly/1PQ6u0t. The Apache Software Foundation also has done a bunch of work in this space. Read more about that at cordova.apache.org. I focused on Windows Phone here because the code is straightforward, easy to debug and simple to explain.

DocumentDB at the Portal

In this next section, I'll build on the previous work, wherein I created a DocumentDB data store for the city and temperature data. To find your DocumentDB database (TemperatureDB), simply click on the menu item, Browse, then select DocumentDB Accounts. You could also pin your database as a tile on the main page by right-clicking, making the main Azure Portal page a dashboard. You can find a nice summary of using the portal tooling to interact with your DocumentDB data store at bit.ly/112L4X1.

One of the really useful features at the new portal is the ability to query and view the actual data in DocumentDB, allowing you to see the city and temperature data in its native JSON format. This dramatically simplifies your ability to easily alter the Node.js Web server and to build out any necessary queries.

Build Out the Node.js Web Server

One of the first things you'll need to do when creating your Node.js Web server is connect to the DocumentDB data store. You'll find the needed connection strings on the Azure Portal. To get to the connection strings from the portal, select the TemperatureDB DocumentDB store, then Click on All Settings, followed by Keys.

From there, you'll need two pieces of information. The first is the URI to the DocumentDB data store. The second is the information security key (called the Primary Key at the portal), which lets you enforce secure access from the Node.js Web server. You'll see the connection and database information in the following code:

```
{
  "HOST"       : "https://temperaturedb.documents.azure.com:443/",
  "AUTH_KEY"   : "secret key from the portal",
  "DATABASE"   : "TemperatureDB",
  "COLLECTION" : "CityTempCollection"
}
```

The Node.js Web server will read the configuration information. The host field in your configuration file will be different, because all URIs are globally unique. The authorization key is also unique.

Figure 1 **The Built Out Node.js Web Server**

```
// +----------------------------+
// |        Section 1           |
// +----------------------------+
var http = require('http');
var port = process.env.port || 1337;
var DocumentDBClient = require('documentdb').DocumentClient;
var nconf = require('nconf');
// +----------------------------+
// |        Section 2           |
// +----------------------------+
// Tell nconf which config file to use
nconf.env();
nconf.file({ file: 'config.json' });
// Read the configuration data
var host = nconf.get("HOST");
var authKey = nconf.get("AUTH_KEY");
var databaseId = nconf.get("DATABASE");
var collectionId = nconf.get("COLLECTION");
// +----------------------------+
// |        Section 3           |
// +----------------------------+
var client = new DocumentDBClient(host, { masterKey: authKey });
// +----------------------------+
// |        Section 4           |
// +----------------------------+
http.createServer(function (req, res) {
  // Before you can query for Items in the document store, you need to ensure you
  // have a database with a collection, then use the collection
  // to read the documents.
  readOrCreateDatabase(function (database) {
    readOrCreateCollection(database, function (collection) {
      // Perform a query to retrieve data and display
      listItems(collection, function (items) {
        var userString = JSON.stringify(items);
        var headers = {
          'Content-Type': 'application/json',
          'Content-Length': userString.length
        };
        res.write(userString);
        res.end();
      });
    });
  });
}).listen(8124,'localhost');  // 8124 seemed to be the
                              // port number that worked
                              // from my development machine.
// +----------------------------+
// |        Section 5           |
// +----------------------------+
// If the database does not exist, then create it, or return the database object.
// Use queryDatabases to check if a database with this name already exists. If you
// can't find one, then go ahead and use createDatabase to create a new database
// with the supplied identifier (from the configuration file) on the endpoint
// specified (also from the configuration file).
var readOrCreateDatabase = function (callback) {
  client.queryDatabases('SELECT * FROM root r WHERE r.id="' + databaseId +
    '"').toArray(function (err, results) {
    console.log('readOrCreateDatabase');

    if (err) {
      // Some error occured, rethrow up
      throw (err);
    }
    if (!err && results.length === 0) {
      // No error occured, but there were no results returned,
      // indicating no database exists matching the query.
      client.createDatabase({ id: databaseId }, function (err, createdDatabase) {
        console.log('client.createDatabase');
        callback(createdDatabase);
      });
    } else {
      // we found a database
      console.log('found a database');
      callback(results[0]);
    }
  });
};
// +----------------------------+
// |        Section 6           |
// +----------------------------+
// If the collection does not exist for the database provided, create it,
// or return the collection object. As with readOrCreateDatabase, this method
// first tried to find a collection with the supplied identifier. If one exists,
// it is returned and if one does not exist it is created for you.
var readOrCreateCollection = function (database, callback) {
  client.queryCollections(database._self, 'SELECT * FROM root r WHERE r.id="' +
    collectionId + '"').toArray(function (err, results) {
    console.log('readOrCreateCollection');
    if (err) {
      // Some error occured, rethrow up
      throw (err);
    }
    if (!err && results.length === 0) {
      // No error occured, but there were no results returned, indicating no
      // collection exists in the provided database matching the query.
      client.createCollection(database._self, { id: collectionId },
        function (err, createdCollection) {
        console.log('client.createCollection');
        callback(createdCollection);
      });
    } else {
      // Found a collection
      console.log('found a collection');
      callback(results[0]);
    }
  });
};
// +----------------------------+
// |        Section 7           |
// +----------------------------+
// Query the provided collection for all non-complete items.
// Use queryDocuments to look for all documents in the collection that are
// not yet complete, or where completed = false. It uses the DocumentDB query
// grammar, which is based on ANSI - SQL to demonstrate this familiar, yet
// powerful querying capability.
var listItems = function (collection, callback) {
  client.queryDocuments(collection._self, 'SELECT c.City,
    c.Temperatures FROM c where c.id="WACO- TX"').toArray(function (err, docs) {
    console.log('called listItems');
    if (err) {
      throw (err);
    }

    callback(docs);
  });
}
```

In previous articles, the temperature database was named TemperatureDB. Each database can have multiple collections, but in this case there's just one called the CityTemperature collection. A collection is nothing more than a list of documents. In this data model, a single document is a city with 12 months of temperature data.

As you dive into the details of the code for the Node.js Web server, you can take advantage of the extensive ecosystem of add-on libraries for Node.js. You'll use two libraries (called Node Packages) for this project. The first package is for DocumentDB functionality. The command to install the DocumentDB package is: npm install documentdb. The second package is for reading the configuration file: npm install nconf. These packages provide additional capabilities missing in the default installation of Node.js. You can find a more extensive tutorial about building out a Node.js application for DocumentDB in the Azure documentation at bit.ly/1E7j5Wg.

There are seven sections in the Node.js Web server, as shown in **Figure 1**. Section 1 covers connections to some of the installed packages so you'll have access to them later in the code. Section 1 also defines the default port to which the mobile client will connect. When deployed to Azure, the port number is managed by Azure, hence the process.env.port construct.

Section 2 reads from the config.json file, which contains the connection information, including the database and document collection. It always makes sense to take string literals relating to connection information and place them separately in a configuration file.

Section 3 is the client connection object you'll use to interact with DocumentDB. The connection is passed to the constructor for DocumentDBClient.

Section 4 represents the code executed once the mobile client connects to the Node.js Web Server application. The createServer is a core primitive for Node.JS applications, and involves a number of concepts around the event loop and processing HTTP requests. You can read more about that construct (bit.ly/1FcNq1E).

This represents the high-level entry point for clients connecting to the Node.js Web server. It's also responsible for calling other pieces of Node.js code that retrieves JSON data from DocumentDB. Once the data is retrieved, it will package it as a JSON-based payload, and deliver it to a mobile client. It leverages the request and response objects, which are parameters to the createServer function, (http.createServer(function (req, res)...).

Section 5 begins the DocumentDB query processing. Your DocumentDB data store can contain multiple databases. The purpose of Section 5 is to narrow down the data at the DocumentDB URI and point to a specific database. In this case, that's TemperatureDB. You'll also see some extra code that isn't directly used, but is there purely for educational purposes. You'll also notice some code to create a database if one does not exist. Much of the logic in Section 5 and beyond is based on the DocumentDB npm package installed earlier.

Section 6 represents the next step in the data retrieval process. The code here is automatically called as a result of the code in Section 5. Section 6 further narrows the data, drilling down to the document collection using the database established (TemperatureDB) in Section 5. You'll note the select statement that includes a where clause for the CityTemperature collection. That includes some code to create a collection if one does not exist.

Section 7 represents the final query executed before data is returned to the mobile client. To keep things simple, the query is hardcoded to return temperature data for the city of Waco, Texas. In a realistic scenario, the mobile client would pass in the city (based on user input or possibly the device's location). The Node.js Web server would then parse the city passed in and append it to the Where clause in Section 7.

The Node.js Web server is now complete and ready to execute. Once it's up and running it, it will wait indefinitely for client requests from the mobile device. You'll run the Node.js Web server locally on your development machine. At this point, it makes sense to use Fiddler to start testing the Node.js Web server. Fiddler lets you issue HTTP requests (in this case, GET) to the Web service and view the response. Validating behavior in Fiddler can help you resolve any issues before building out the mobile client.

Now you're ready to build out the mobile client, which involves two basic architectural components—the XAML UI and the CS Code-Behind (where the programming logic lives). The code shown in **Figure 2** represents the markup for the visual interface on the mobile client.

Notice that it incorporates the WinRTXamlToolkit, which you can find on CodePlex at bit.ly/1PQdXw0. This toolkit includes a number of interesting controls. The one you'll use is the chart control. To chart data, simply build up a name/value collection and attach it to the control. In this case, you'll build out a name/value collection of rainfall data for each month in a given city.

Before presenting the final solution, there are some caveats. One could dispute using a native Windows Phone application, in favor of a more Web-based approach.

The mobile client built out here takes a number of shortcuts to come up with the bare minimum functionality. For example, the bar chart will appear immediately once you run the Windows Phone client. That's because there's a Web request to the Node.js Web server from the OnNavigatedTo event. This automatically executes once the Windows Phone client starts. You can see this in Section 1 of the mobile client code shown in **Figure 3**.

Figure 2 **The XAML Markup for Mobile Client Main Screen Bar Chart**

```
<Page
  x:Class="CityTempApp.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:CityTempApp"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  xmlns:charting="using:WinRTXamlToolkit.Controls.DataVisualization.Charting"

  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

  <Grid>
    <!-- WinRTXamlToolkit chart control -->
    <charting:Chart
      x:Name="BarChart"
      Title=""
      Margin="5,0">
      <charting:BarSeries
        Title="Rain (in)"
        IndependentValueBinding="{Binding Name}"
        DependentValueBinding="{Binding Value}"
        IsSelectionEnabled="True"/>
    </charting:Chart>

  </Grid>
</Page>
```

SEE THE WORLD AS A DATABASE
ADO.NET ▪ JDBC ▪ ODBC ▪ SQL SSIS ▪ ODATA
MYSQL ▪ EXCEL ▪ POWERSHELL
Visual Studio
Java
ODBC
SQL Server
Excel
BizTalk
MySQL
OData

Work With Relational Data, Not Complex APIs or Services

Whether you are a developer using ADO.NET, JDBC, OData, or MySQL, or a systems integrator working with SQL Server or Biztalk, or even an information worker familiar with ODBC or Excel – our products give you bi-directional access to live data through easy-to-use technologies that you are already familiar with.   If you can connect to a database, then you will already know how to connect to Salesforce, SAP, SharePoint, Dynamics CRM, Google Apps, QuickBooks, and much more!

Give RSSBus a try today and see what mean:

visit us online at www.rssbus.com to learn more or download a free trial.

rssbus

INTEGRATION YOUR WAY

Copyright © 2013 RSSBus, Inc. All rights reserved. All trademarks and registered trademarks are the property of their respective owners.

Figure 3 **Code for the Mobile Client**

```
public sealed partial class MainPage : Page
{
  public MainPage()
  {
    this.InitializeComponent();
    this.NavigationCacheMode = NavigationCacheMode.Required;
  }
  // +----------------------------+
  // |       Section 1            |
  // +----------------------------+
  protected override void OnNavigatedTo(NavigationEventArgs e)
  {
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(
      "http://localhost:8124");
    request.BeginGetResponse(MyCallBack, request);
  }
  // +----------------------------+
  // |       Section 2            |
  // +----------------------------+
  async void MyCallBack(IAsyncResult result)
  {
    HttpWebRequest request = result.AsyncState as HttpWebRequest;
    if (request != null)
    {
      try
      {
        WebResponse response = request.EndGetResponse(result);
        Stream stream = response.GetResponseStream();
        StreamReader reader = new StreamReader(stream);
        JsonSerializer serializer = new JsonSerializer();
        // +----------------------------+
        // |       Section 3            |
        // +----------------------------+
        // Data structures coming back from Node
        List<CityTemp> cityTemp = (List<CityTemp>)serializer.Deserialize(
          reader, typeof(List<CityTemp>));
        // Data structure suitable for the chart control on the phone
        List<NameValueItem> items = new List<NameValueItem>();
        string[] months = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
        for (int i = 11; i >= 0; i-- )
        {
          items.Add(new NameValueItem { Name = months[i], Value =
            cityTemp[0].Temperatures[i] });
        }
        // +----------------------------+
        // |       Section 4            |
        // +----------------------------+
        // Provide bar chart the data
        await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
          this.BarChart.Title = cityTemp[0].City + ", 2014";
          ((BarSeries)this.BarChart.Series[0]).ItemsSource = items;
        });
      }
      catch (WebException e)
      {
        return;
      }
    }
  }
}
// +----------------------------+
// |       Section 5            |
// +----------------------------+
// Data structures coming back from Node
public class CityTemp
{
  public string City { get; set;  }
  public List<double> Temperatures { get; set; }
}
// Data structure suitable for the chart control on the phone
public class NameValueItem
{
  public string Name { get; set; }
  public double Value { get; set; }
}
```

Also in Section 1, you'll notice it's connecting to the Node.js Web server running on localhost. Obviously, you would specify a different endpoint if you're hosting your Node.js Web server in the public cloud, such as Azure Web sites. After issuing the Web request, you set up the callback using BeginGetResponse. Web requests are asynchronous, so the code sets up a callback (MyCallBack). This leads to Section 2, where the data is retrieved, processed and loaded into a chart control.

The callback in Section 2 for the asynchronous Web request outlined in Section 1 processes the payload sent back by the Web service. The code processes the Web response, serializing the data, which is in JSON format. Section 3 is about transforming that JSON data into two separate data structures defined in Section 5. The goal is to create a list of name/value arrays or lists. The NameValueItem class is the structure the chart control needs.

Section 4 is about using the GUI thread to assign the name/value list to the chart control. You can see the item assignment in the items source collection. The await this.Dispatcher.RunAsync syntax leverages the GUI thread for updating visual controls. The code won't work properly if you try to update the visual interface using the same thread as the one processing the data and making the Web request.

Now you can run the mobile client. You might be missing some of the temperature data, though, so all the bar controls might not appear.

Wrapping Up

This concludes the three-part series, where I set out to show an end-to-end IoT scenario—from data ingestion to persisting the data to visualizing the data. I started this series by ingesting the data using a C program running under Ubuntu, as an effort to simulate the code you'd need to run under a Raspberry Pi device. You can insert data captured from a temperature sensor into Azure Event Hubs. However, data stored here is ephemeral and you need to move it to a more permanent store.

That brought you to the second article, in which a background process takes the data from Event Hubs and moves it to both Azure SQL Database and DocumentDB. Then this final installment exposed this permanently stored data to mobile devices using a middle tier running Node.js.

There are many potential extensions and improvements that can be performed. For example, one area you might explore is the notion of machine learning and analytics. The bar chart visualization answers the basic question, "What happened?" A more interesting question might be, "What will happen?" In other words, can you then predict future rainfall? The ultimate question could be, "What should I do today based on future predictions?" ■

BRUNO TERKALY *is a principal software engineer at Microsoft with the objective of enabling development of industry-leading applications and services across devices. He's responsible for driving the top cloud and mobile opportunities across the United States and beyond from a technology-enablement perspective. He helps partners bring their applications to market by providing architectural guidance and deep technical engagement during the ISV's evaluation, development and deployment. Terkaly also works closely with the cloud and mobile engineering groups, providing feedback and influencing the roadmap.*

THANKS to the following Microsoft technical experts for reviewing this article: *Leland Holmquest, David Magnotti and Nick Trogh*

Clean Data Made Easy

One of the most important steps to ensuring business success is to keep your database clean. We have easy-to-integrate data quality tools to do just that.

Verify Global Contacts

Standardize, verify and update name, address, phone, and email info for 240+ countries.

Get the Dupes Out

Detect and merge duplicate or similar records to create a single view of the customer.

Enrich Your Data

Add geographic, demographic, IP location, and property/mortgage data for better insights and business decisions.

Clean it Your Way

Pick only the tools you need: Cloud or on-premise APIs; Microsoft®; Oracle®; Pentaho; Salesforce®; and more!

Free Trials Available!
www.MelissaData.com/easy

Germany
www.MelissaData.de

United Kingdom
www.MelissaData.co.uk

India
www.MelissaData.in

Australia
www.MelissaData.com.au

MELISSA DATA®
Your Partner in Data Quality
www.MelissaData.com | 1-800-MELISSA

Fill-up on real-world, practical information and training on the Microsoft Platform as Visual Studio Live! returns to warm, sunny Orlando for the conference more developers rely on to code with industry experts and successfully navigate the .NET highway.

Connect with Visual Studio Live!

twitter.com/vslive
@VSLive

facebook.com
Search "VSLive"

linkedin.com – Join the
"Visual Studio Live" group!

EVENT PARTNERS
Microsoft Magenic

PLATINUM SPONSORS
GRIDSTORE pluralsight

GOLD SPONSOR
GitHub

SUPPORTED BY
Visual Studio msdn magazine

5 *Great Conferences* 1 *Great Price*

Whether you are an

- Engineer
- Developer
- Programmer
- Software Architect
- Software Designer

You will walk away from this event having expanded your .NET skills and the ability to build better applications.

REGISTER BY AUGUST 12 AND **SAVE $500!**

Use promo code VSLORL2 by August 12

Scan the QR code to register or for more event details.

Take the Tour

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

Visual Studio Live! Orlando is part of Live! 360, the Ultimate Education Destination. This means you'll have access to four (4) other co-located events at no additional cost:

SharePoint LIVE!
TRAINING FOR COLLABORATION

SQL Server LIVE!
SQL SERVER FOR MODERN DEVELOPERS

Modern Apps LIVE!
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

Five (5) events and hundreds of sessions to choose from - mix and match sessions to create your own, custom event line-up - it's like no other conference available today!

Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

VSLIVE.COM/ORLANDO

Using STL Strings at Win32 API Boundaries

Giovanni Dicanio

The Win32 API exposes several features using a pure-C interface. This means there are no C++ string classes available natively to exchange text at the Win32 API boundaries. Instead, raw C-style character pointers are used. For example, the Win32 SetWindowText function has the following prototype (from the associated MSDN documentation at bit.ly/1Fkb5lw):

```
BOOL WINAPI SetWindowText(
  HWND hWnd,
  LPCTSTR lpString
);
```

The string parameter is expressed in the form of LPCTSTR, which is equivalent to const TCHAR*. In Unicode builds (which have been the default since Visual Studio 2005 and should be used in modern Windows C++ applications), the TCHAR typedef corresponds to wchar_t, so the prototype of SetWindowText reads as:

```
BOOL WINAPI SetWindowText(
  HWND hWnd,
  const wchar_t* lpString
);
```

This article discusses:

- The Win32 API
- Passing standard STL strings as input and out parameters
- Handling a potential race condition

Technologies discussed:

C++, Standard Template Library, Visual Studio, Win32 API

So, basically, the input string is passed as a constant (that is, read-only) wchar_t character pointer, with the assumption that the string pointed to is NUL-terminated, in a classical, pure-C style. This is a typical pattern for input string parameters passed at the Win32 API boundaries.

Of course, using C++ instead of pure C is an extremely productive option for developing user-mode Windows code, and Windows applications in particular.

On the other side, output strings at the Win32 API boundaries are typically represented using a couple of pieces of information: a pointer to a destination buffer, allocated by the caller, and a size parameter, representing the total size of the caller-provided buffer. An example is the GetWindowText function (bit.ly/1bAMkpA):

```
int WINAPI GetWindowText(
  HWND hWnd,
  LPTSTR lpString,
  int nMaxCount
);
```

In this case, the information related to the destination string buffer (the "output" string parameter) is stored in the last two parameters: lpString and nMaxCount. The former is a pointer to the destination string buffer (represented using the LPTSTR Win32 typedef, which translates to TCHAR*, or wchar_t* in Unicode builds). The latter, nMaxCount, represents the total size of the destination string buffer, in wchar_ts; note that this value includes the terminating NUL character (don't forget that those C-style strings are NUL-terminated character arrays).

In fact, in general, the use of C++ raises the semantic level of the code and increases programmer productivity, without a negative impact on application performance.

Of course, using C++ instead of pure C is an extremely productive option for developing user-mode Windows code, and Windows applications in particular. In fact, in general, the use of C++ raises the semantic level of the code and increases programmer productivity, without a negative impact on application performance. In particular, using convenient C++ string classes is much better (easier, more productive, less bug-prone) than dealing with raw, C-like, NUL-terminated character arrays.

So the question now becomes: What kind of C++ string classes can be used to interact with the Win32 API layer, which natively exposes a pure-C interface?

Active Template Library (ATL)/Microsoft Foundation Class (MFC) Library CString Well, the ATL/MFC CString class is an option. CString is very well integrated with C++ Windows frameworks such as ATL, MFC and Windows Template Library (WTL), which simplify Win32 programming using C++. So it makes sense to use CString to represent strings at the Win32 API platform-specific layer of C++ Windows applications if you use those frameworks. Moreover, CString offers convenient Windows platform-specific features, like being able to load strings from resources, and so forth; those are platform-dependent features that a cross-platform standard library like the Standard Template Library (STL) simply can't offer, by definition. So, for example, if you need to design and implement a new C++ class, derived from some existing ATL or MFC class, definitely consider using CString to represent strings.

Standard STL Strings However, there are cases where it's better to use a standard string class at the interface of custom-designed C++ classes that make up Windows applications. For example, you may want to abstract away the Win32 API layer as soon as possible in your C++ code, preferring the use of an STL string class instead of Windows-specific classes like CString at the public interface of custom-designed C++ classes. So, let's consider the case of text stored in STL string classes. At this point, you need to pass those STL strings across Win32 API boundaries (which expose a pure-C interface, as discussed in the beginning of this article). With ATL, WTL and MFC, the framework implements the "glue" code between the Win32 C interface layer and CString, hiding it under the hood, but this convenience isn't available with STL strings.

For the purpose of this article, let's assume the strings are stored in Unicode UTF-16 format, which is the default Unicode encoding for Windows APIs. In fact, if those strings used another format (such as Unicode UTF-8), those could be converted to UTF-16 at the Win32 API boundary, satisfying the aforementioned requirement of this article. For these conversions, the Win32 MultiByteToWideChar and WideCharToMultiByte functions could be used: The former can be called to convert from a Unicode UTF-8-encoded ("multi-byte") string to a Unicode UTF-16 ("wide") string; the latter can be used for the opposite conversion.

In Visual C++, the std::wstring type is well-suited to represent a Unicode UTF-16 string, because its underlying character type is wchar_t, which has a size of 16 bits in Visual C++, the exact size of a UTF-16 code unit. Note that on other platforms, such as GCC Linux, a wchar_t is 32 bits, so a std::wstring on those platforms would be well-suited to represent Unicode UTF-32-encoded text. To remove this ambiguity, a new standard string type was introduced in C++11: std::u16string. This is a specialization of the std::basic_string class with elements of type char16_t, that is, 16-bit character units.

The Input String Case

If a Win32 API expects a PCWSTR (or LPCWSTR in older terminology), that is, a const wchar_t* NUL-terminated C-style input string parameter, simply calling the std::wstring::c_str method will be just fine. In fact, this method returns a pointer to a read-only NUL-terminated C-style string.

CString is very well integrated with C++ Windows frameworks such as ATL, WTL and MFC, which simplify Win32 programming using C++.

For example, to set the text of a window's titlebar or the text of a control using the content stored in a std::wstring, the SetWindowText Win32 API can be called like this:

```
std::wstring text;
::SetWindowText(hWnd, text.c_str());
```

Note that, while the ATL/MFC CString offers an implicit conversion to a raw character const pointer (const TCHAR*, which is equivalent to const wchar_t* in modern Unicode builds), STL strings do not offer such an implicit conversion. Instead, you must make an explicit call to the STL string's c_str method. There's a common understanding in modern C++ that implicit conversions

Figure 1 **Reading Strings in Place Using std::wstring**

```
// Get the length of the text string
// (Note: +1 to consider the terminating NUL)
const int bufferLength = ::GetWindowTextLength(hWnd) + 1;

// Allocate string of proper size
std::wstring text;
text.resize(bufferLength);

// Get the text of the specified control
// Note that the address of the internal string buffer
// can be obtained with the &text[0] syntax
::GetWindowText(hWnd, &text[0], bufferLength);

// Resize down the string to avoid bogus double-NUL-terminated strings
text.resize(bufferLength - 1);
```

tend to not be a good thing, so the designers of STL string classes opted for an explicitly callable c_str method. (You'll find a related discussion on the lack of implicit conversion in modern STL smart pointers in the blog post at bit.ly/1d9AGT4.)

The Output String Case

Things become a little bit more complicated with output strings. The usual pattern consists of first calling a Win32 API to get the size of the destination buffer for the output string. This may or may not include the terminating NUL; the documentation of the particular Win32 API must be read for that purpose.

Then, a buffer of proper size is allocated dynamically by the caller. The size of that buffer is the size determined in the previous step.

And, finally, another call is made to a Win32 API to read the actual string content into the caller-allocated buffer.

For example, to retrieve the text of a control, the GetWindowTextLength API can be invoked to get the length, in wchar_ts, of the text string. (Note that, in this case, the returned length does *not* include the terminating NUL.)

Then, a string buffer can be allocated using that length. An option here could be to use a std::vector<wchar_t> to manage the string buffer, for example:

```
// Get the length of the text string
// (Note: +1 to consider the terminating NUL)
const int bufferLength = ::GetWindowTextLength(hWnd) + 1;

// Allocate string buffer of proper size
std::vector<wchar_t> buffer(bufferLength);
```

Note that this is simpler than using a raw "new wchar_t[bufferLength]" call, because that would require properly releasing the buffer with a call to delete[] (and forgetting to do that would cause a memory leak). Using std::vector is just simpler, even if using vector has a small overhead compared to a raw new[] call. In fact, in that case the std::vector's destructor would automatically delete the allocated buffer.

This also helps in building exception-safe C++ code: If an exception is thrown somewhere in the code, the std::vector destructor would be automatically called. Instead, a buffer dynamically allocated with new[], whose pointer is stored in a raw owning pointer, would be leaked.

Another option, considered as an alternative to std::vector, might be the use of std::unique_ptr, in particular, std::unique_ptr< wchar_t[] >. This option has the automatic destruction (and exception-safety) of std::vector (thanks to std::unique_ptr's destructor), as well as less overhead than std::vector, because std::unique_ptr is a very tiny C++ wrapper around a raw owning pointer. Basically, unique_ptr is an owning pointer protected within safe RAII boundaries. RAII (bit.ly/1AbSa6k) is a very common C++ programming idiom. If you're unfamiliar with it, just think of RAII as an implementation technique that automatically calls delete[] on the wrapped pointer—for example, in unique_ptr's destructor—releasing the associated resources and preventing memory leaks (and resource leaks, in general).

With unique_ptr, the code might look something like this:

```
// Allocate string buffer using std::unique_ptr
std::unique_ptr< wchar_t[] > buffer(new wchar_t[bufferLength]);
```

Or, using std::make_unique (available since C++14 and implemented in Visual Studio 2013):

```
auto buffer = std::make_unique< wchar_t[] >(bufferLength);
```

Then, once a buffer of proper size is allocated and ready for use, the GetWindowText API can be called, passing a pointer to that string buffer. To get a pointer to the beginning of the raw buffer managed by the std::vector, the std::vector::data method (bit.ly/1I3ytEA) can be used, like so:

```
// Get the text of the specified control
::GetWindowText(hWnd, buffer.data(), bufferLength);
```

With unique_ptr, its get method could be called instead:

```
// Get the text of the specified control (buffer is a std::unique_ptr)
::GetWindowText(hWnd, buffer.get(), bufferLength);
```

And, finally, the text of the control can be deep copied from the temporary buffer into a std::wstring instance:

```
std::wstring text(buffer.data()); // When buffer is a std::vector<wchar_t>

std::wstring text(buffer.get()); // When buffer is a std::unique_ptr<wchar_t[]>
```

In the preceding code snippet, I used a constructor overload of wstring, taking a constant raw wchar_t pointer to a NUL-terminated input string. This works just fine, because the called Win32 API will insert a NUL terminator in the destination string buffer provided by the caller.

As a form of slight optimization, if the length of the string (in wchar_ts) is known, a wstring constructor overload taking a pointer and a string character count parameter could be used instead. In this case, the string length is provided at the call site, and the wstring constructor doesn't need to find it out (typically with an O(N) operation, like calling wcslen in a Visual C++ implementation).

Figure 2 **Sample Coding Pattern to Handle a Potential Race Condition in Getting Strings**

```
LONG error = ERROR_MORE_DATA;
std::unique_ptr< wchar_t[] > buffer;
DWORD bufferLength = /* Some initial reasonable length for the string buffer */;
while (error == ERROR_MORE_DATA)
{
  // Create a buffer with bufferLength size (measured in wchar_ts)
  buffer = std::make_unique<wchar_t[]>(bufferLength);

  // Call the desired Win32 API
  error = ::SomeApiCall(param1, param2, ..., buffer.get(), &bufferLength);
}

if (error != ERROR_SUCCESS)
{
  // Some error occurred
  // Handle it e.g. throwing an exception
}

// All right!
// Continue processing
// Build a std::wstring with the NUL-terminated text
// previously stored in the buffer
std::wstring text(buffer.get());
```

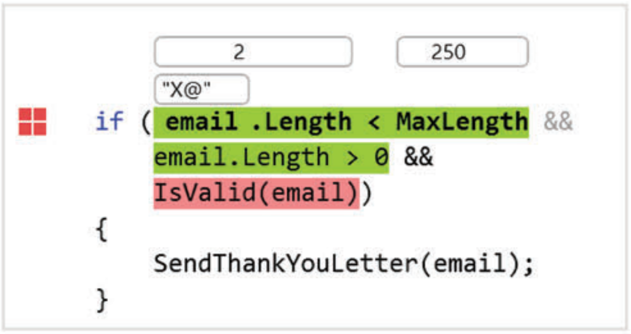
OzCode Your Road to Magical Debugging

Try v2.0 Beta Now!

Debug Like a Boss

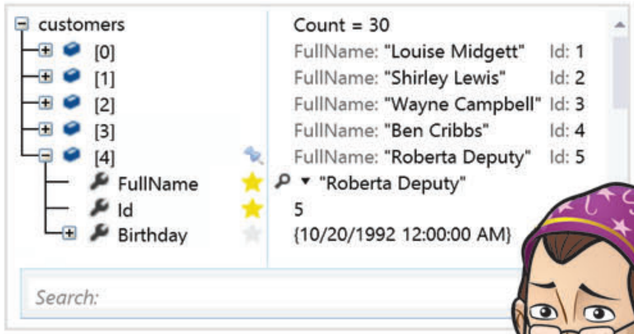
Enhance Your C# Debugging Experience

Simplify


```
                 2                 250
            "X@"
     if ( email .Length < MaxLength &&
          email.Length > 0 &&
          IsValid(email))
     {
          SendThankYouLetter(email);
     }
```

Make sense of complex expressions and predict the next result

Reveal


```
customers                 Count = 30
  [0]                     FullName: "Louise Midgett"   Id: 1
  [1]                     FullName: "Shirley Lewis"    Id: 2
  [2]                     FullName: "Wayne Campbell"   Id: 3
  [3]                     FullName: "Ben Cribbs"       Id: 4
  [4]                     FullName: "Roberta Deputy"   Id: 5
    FullName              "Roberta Deputy"
    Id                    5
    Birthday              {10/20/1992 12:00:00 AM}
Search:
```

Visualize objects and focus on data that actually matters

Search
Find a needle in a haystack of data

Exception Trail
Easily navigate inner exceptions

Compare
View differences between objects

Quick Actions
Automate your everyday debugging

Trace
Log, view and filter debug messages

Show All Instances
Track down an object in memory

Supports Visual Studio 2010, 2012, 2013

Visual Studio Partner

Get an exclusive DISCOUNT

oz-code.com | info@oz-code.com

Powered by CodeValue

A Shortcut for the Output Case: Working in Place with std::wstring

With regard to the technique of allocating a temporary string buffer using an std::vector (or an std::unique_ptr) and then deep copying it into a std::wstring, you could take a shortcut.

Basically, an instance of std::wstring could be used *directly* as a destination buffer to pass to Win32 APIs.

In fact, std::wstring has a resize method that can be used to build a string of proper size. Note that in this case, you don't care about the actual initial content of the resized string, because it will be overwritten by the invoked Win32 API. **Figure 1** contains a sample code snippet showing how to read strings in place using std::wstring.

I'd like to clarify a few points related to the code in **Figure 1**.

Getting Write Access to the Internal String Buffer First, consider the GetWindowText call:

```
::GetWindowText(hWnd, &text[0], bufferLength);
```

A C++ programmer might be tempted to use the std::wstring::-data method to access the internal string content, via a pointer to be passed to the GetWindowText call. But wstring::data returns a const pointer, which wouldn't allow the content of the internal string buffer to be modified. And because GetWindowText expects write access to the content of the wstring, that call wouldn't compile. So, an alternative is to use the &text[0] syntax to get the address of the beginning of the internal string buffer, to be passed as an output (that is, modifiable) string to the desired Win32 API.

Compared to the previous approach, this technique is more efficient because there's no temporary std::vector, with a buffer first allocated, then deep copied into a std::wstring and, finally, discarded. In fact, in this case, the code just operates in place in a std::wstring instance.

Avoiding Bogus Double-NUL-Terminated Strings Pay attention to the last line of code in **Figure 1**:

```
// Resize down the string to avoid bogus double-NUL-terminated strings
text.resize(bufferLength - 1);
```

With the initial wstring::resize call (text.resize(bufferLength);, without the "-1" correction), enough room is allocated in the internal wstring buffer to allow the GetWindowText Win32 API to scribble in its NUL terminator. However, in addition to this NUL terminator written by GetWindowText, std::wstring implicitly provides another NUL terminator. So, the resulting string ends up as a double-NUL-terminated string: the NUL terminator written by GetWindowText, and the NUL terminator automatically added by wstring.

To fix this wrong double-NUL-terminated string, the wstring instance can be *resized down* to chop the NUL terminator added by the Win32 API off, leaving only wstring's NUL terminator. This is the purpose of the text.resize(bufferLength-1) call.

Handling a Race Condition

Before concluding this article, it's worth discussing how to handle a potential race condition that may arise with some APIs. For example, suppose you have code that's reading some string value from the Windows Registry. Following the pattern showed in the previous section, a C++ programmer would first call the RegQueryValueEx function to get the length of the string value. Then, a buffer for the string would be allocated, and finally the RegQueryValueEx would be called a second time, to read the actual string value into the buffer created in the previous step.

The race condition that could arise in this case is another process modifying the string value between the two RegQueryValueEx calls. The string length returned by the first call could be a meaningless value, unrelated to the new string value written in the Registry by the other process. So, the second call to RegQueryValueEx would read the new string in a buffer allocated with a wrong size.

To fix that bug, you can use a coding pattern like the one in **Figure 2**.

The use of the while loop in **Figure 2** ensures that the string is read in a buffer of proper length, because each time ERROR_MORE_DATA is returned, a new buffer is allocated with the proper bufferLength value until the API call succeeds (returning ERROR_SUCCESS) or fails for a reason other than providing an insufficiently sized buffer.

In fact, std::wstring has a resize method that can be used to build a string of proper size.

Note that the code snippet in **Figure 2** is just example skeleton code; other Win32 APIs could use different error codes related to an insufficient buffer provided by the caller, for example, the ERROR_INSUFFICIENT_BUFFER code.

Wrapping Up

While the use of CString at the Win32 API boundary—with the help of frameworks like ATL/WTL and MFC—hides the mechanics of interoperation with the Win32 pure-C-interface layer, when using STL strings the C++ programmer must pay attention to certain details. In this article, I discussed some coding patterns for the interoperation of the STL wstring class and Win32 pure-C interface functions. In the input case, calling wstring's c_str method is just fine to pass an input string at the Win32 C interface boundary, in the form of a simple constant (read-only) NUL-terminated string character pointer. For output strings, a temporary string buffer must be allocated by the caller. This can be achieved using either the std::vector STL class or, with slightly less overhead, the STL std::unique_ptr smart pointer template class. Another option is to use the wstring::resize method to allocate some room inside the string instance as a destination buffer for Win32 API functions. In this case, it's important to specify enough space to allow the invoked Win32 API to scribble in its NUL terminator, and then resize down to chop that off and leave only wstring's NUL terminator. Finally, I covered a potential race condition, and presented a sample coding pattern to solve that race condition. ■

Giovanni Dicanio *is a computer programmer specializing in C++ and the Windows OS, a Pluralsight author and a Visual C++ MVP. Besides programming and course authoring, he enjoys helping others on forums and communities devoted to C++, and can be contacted at giovanni.dicanio@gmail.com.*

Thanks to the following technical experts for reviewing this article:
David Cravey (GlobalSCAPE) and Stephan T. Lavavej (Microsoft)

TechMentor 2016 WILL NOT be returning to Redmond – DON'T MISS this unique opportunity!

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

MICROSOFT HEADQUARTERS, REDMOND, WA

AUGUST 3 - 7, 2015

ENGINEERED FOR YOU @ THE SOURCE

Join us August 3 – 7, 2015 for TechMentor 2015: Datacenter Edition, focused entirely on making your datacenter more modern, capable, and manageable through 5 days of immediately usable IT education.

SESSIONS ARE FILLING UP QUICKLY —

REGISTER TODAY!

USE PROMO CODE **TMJUL1**

Scan the QR code to register or for more event details.

THE AGENDA FEATURES:

- 75-minute topic overview breakout sessions
- 3 hour content-rich deep dives
- "Hands on" labs with your own laptop

CONTENT AREAS INCLUDE:

- Windows PowerShell
- Infrastructure Foundations
- Runbook Automation
- Configuration and Service Management
- Datacenter Provisioning and Deployment

EVENT SPONSOR: Microsoft

PLATINUM SPONSOR: GRIDSTORE

GOLD SPONSORS: GOVERLAN PLURALSIGHT

SUPPORTED BY: Redmond Magazine, Redmond Channel Partner, Virtualization Review

PRODUCED BY: 1105 MEDIA

TECHMENTOREVENTS.COM/REDMOND

Introduction to R for C# Programmers

James McCaffrey

The R language is used by data scientists and programmers for statistical computing. In part because of the increasing amounts of data collected by software systems, and the need to analyze that data, R is one of the fastest-growing technologies among my colleagues who use C#. A familiarity with R can be a valuable addition to your technical skill set.

The R language is a GNU project and is free software. R was derived from a language called S (for "statistics"), which was created at Bell Laboratories in the 1970s. There are many excellent online tutorials for R, but most of those tutorials assume you're a university student studying statistics. This article is aimed at helping C# programmers get up to speed with R as quickly as possible.

The best way to see where this article is headed is to take a look at the example R session in **Figure 1**. The example session has two unrelated topics. The first few set of commands show what's called a chi-square test (also called the chi-*squared* test) for a uniform distribution. The second set of commands shows an example of linear regression, which in my opinion is the Hello World technique of statistical computing.

The R Web site is located at r-project.org. The site has links to several mirror sites where you can download and install R. The install is a simple self-extracting executable. R is officially supported on Windows XP and later, and also on most common non-Windows platforms. I've installed R on Windows 7 and Windows 8 machines without any issues. By default the installation process gives you both 32-bit and 64-bit versions.

This article assumes you have at least intermediate C# programming skills (so you can understand the explanations of the similarities and differences between C# and R), but doesn't assume you know anything about R. It presents a C# demo program in its entirety, which is also available from the download file that accompanies this article.

This article discusses:

- An example chi-square test using R
- Linear regression analysis
- A chi-square test of independence
- Graphs with R
- An example chi-square test in C# for comparison

Technologies discussed:

R, C#, Visual Studio

Code download available at:

msdn.microsoft.com/magazine/msdnmag0715

The Chi-Square Test Using R

Looking at **Figure 1**, the first thing to notice is that using R is quite a bit different from using C#. Although it's possible to write R scripts, R is most often used in interactive mode in a command shell. The first R example is an analysis to see if a normal six-sided die is fair or not. When rolled many times, a fair die would be expected to give approximately the same count for each of the six possible results.

The R prompt is indicated by the (>) token in the shell. The first statement typed in **Figure 1** begins with the (#) character, which is the R token to indicate a comment.

The first actual command in the example session is:

```
> observed <- c(20, 28, 12, 32, 22, 36)
```

This creates a vector named observed using the c (for concatenate) function. A vector holds objects with the same data type. A roughly equivalent C# statement would be:

```
var observed = new int[] { 20, 28, 12, 32, 22, 36 };
```

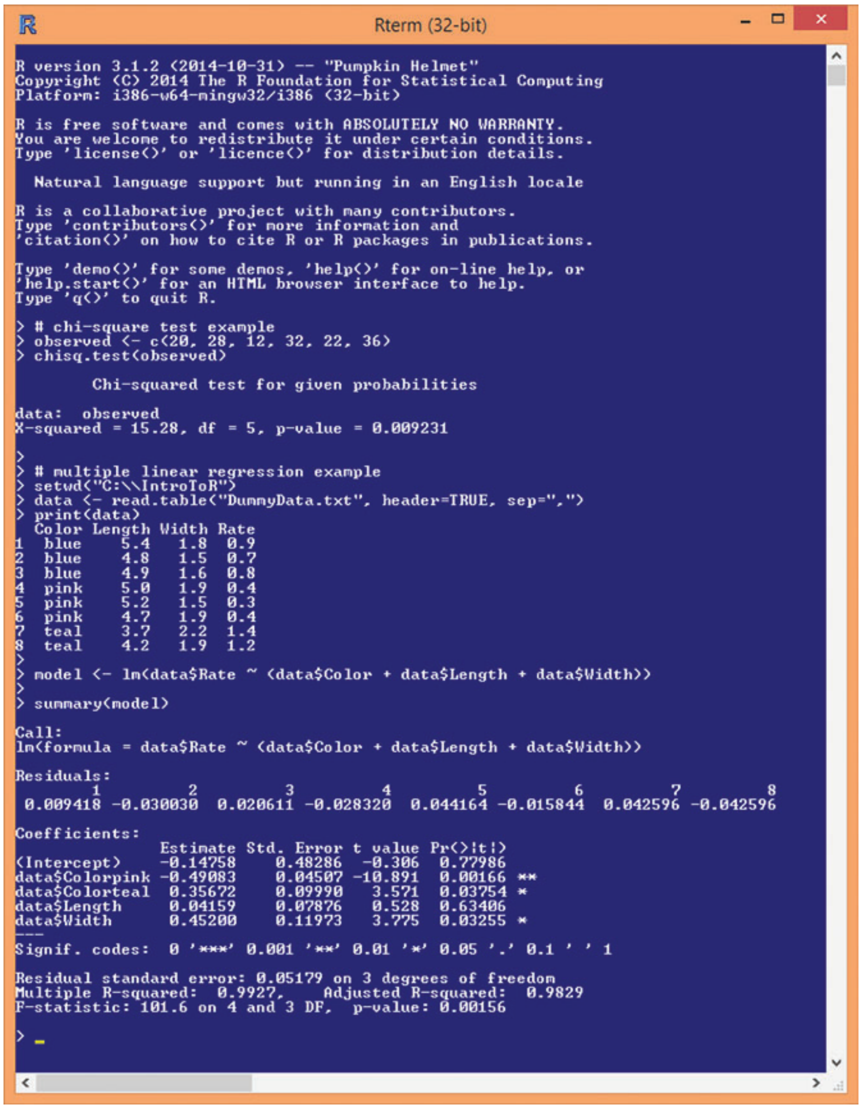
The first value, 20, is the number of times a one-spot occurred, 28 is the number of times a two-spot occurred and so on. The sum of the six counts is 20 + 28 + . . + 36 = 150. You'd expect a fair die to have about 150/6 = 25 counts for each possible outcome. But the number of observed three-spots (12) looks suspiciously low.

After creating the vector, a chi-square test is performed using the chisq.test function:

```
> chisq.test(observed)
```

In R, the dot (.) character is often used rather than the underscore (_) character to create variable and function names that are easier to read. The result of calling the chisq.test function is quite a bit of text:

```
Chi-squared test for given probabilities
data:  observed
X-squared = 15.28, df = 5, p-value = 0.009231
```



```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> # chi-square test example
> observed <- c(20, 28, 12, 32, 22, 36)
> chisq.test(observed)

        Chi-squared test for given probabilities

data:  observed
X-squared = 15.28, df = 5, p-value = 0.009231

>
> # multiple linear regression example
> setwd("C:\\IntroToR")
> data <- read.table("DummyData.txt", header=TRUE, sep=",")
> print(data)
  Color Length Width Rate
1  blue    5.4   1.8  0.9
2  blue    4.8   1.5  0.7
3  blue    4.9   1.6  0.8
4  pink    5.0   1.9  0.4
5  pink    5.2   1.5  0.3
6  pink    4.7   1.9  0.4
7  teal    3.7   2.2  1.4
8  teal    4.2   1.9  1.2
>
> model <- lm(data$Rate ~ (data$Color + data$Length + data$Width))
>
> summary(model)

Call:
lm(formula = data$Rate ~ (data$Color + data$Length + data$Width))

Residuals:
        1         2         3         4         5         6         7         8
 0.009418 -0.030030  0.020611 -0.028320  0.044164 -0.015844  0.042596 -0.042596

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    -0.14758    0.48286  -0.306  0.77986
data$Colorpink -0.49083    0.04507 -10.891  0.00166 **
data$Colorteal  0.35672    0.09990   3.571  0.03754 *
data$Length     0.04159    0.07876   0.528  0.63406
data$Width      0.45200    0.11973   3.775  0.03255 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.05179 on 3 degrees of freedom
Multiple R-squared:  0.9927,    Adjusted R-squared:  0.9829
F-statistic: 101.6 on 4 and 3 DF,  p-value: 0.00156

> _
```

Figure 1 **An Example R Session**

In C# terms, most R functions return a data structure that can be ignored, but also contain many Console.WriteLine equivalent statements that expose output. Notice that it's up to you to decipher the meaning of R output. Later in this article, I'll show you how to create the equivalent chi-square test using raw (no libraries) C# code.

In this example, the "X-squared" value of 15.28 is the calculated chi-squared statistic (the Greek letter chi resembles uppercase X). A value of 0.0 indicates that the observed values are exactly what you'd expect if the die was fair. Larger values of chi-squared indicate an increasing likelihood that the observed counts could not have happened by chance if the die was fair. The df value of 5 is the "degrees of freedom," which is one less than the number of observed values. The df is not too important for this analysis.

The p-value of 0.009231 is the probability that the observed counts could have occurred by chance if the die was fair. Because the p-value is so small, less than 1 percent, you'd conclude that the observed values are very unlikely to have occurred by chance and, therefore, there's statistical evidence that the die in question is most likely biased.

Linear Regression Analysis Using R

The second set of statements in **Figure 1** shows an example of linear regression. Linear regression is a statistical technique used to describe the relationship between a numeric variable (called the dependent variable in statistics) and one or more explanatory variables (called the independent variables) that can be either numeric or categorical. When there's just one independent explanatory/predictor variable, the technique is called simple linear regression. When there are two or more independent variables, as in the demo example, the technique is called multiple linear regression.

Before doing the linear regression analysis, I created an eight-item, comma-delimited text file named DummyData.txt in directory C:\IntroToR with this content:

```
Color,Length,Width,Rate
blue, 5.4, 1.8, 0.9
blue, 4.8, 1.5, 0.7
blue, 4.9, 1.6, 0.8
pink, 5.0, 1.9, 0.4
pink, 5.2, 1.5, 0.3
pink, 4.7, 1.9, 0.4
teal, 3.7, 2.2, 1.4
teal, 4.2, 1.9, 1.2
```

This file is supposed to represent flower data with the color of the flower, the length and width of the petal, and the growth rate. The idea is to predict Rate values (in the last column) from the Color, Length and Width values. After a comment statement, the first three R commands in the linear regression analysis are:

```
> setwd("C:\\IntroToR")
> data <- read.table("DummyData.txt",
  header=TRUE, sep=",")
> print(data)
```

The first command sets the working directory so I wouldn't have to fully qualify the path to the source data file. Instead of using the (\\) token as is common with C#, I could have used (/) as is common on non-Windows platforms.

The second command loads the data into memory in a table object named data. Notice that R uses named parameters. The header parameter tells if the first line is header information (TRUE, or T in shortened form) or not (FALSE or F). R is case-sensitive and you can usually use either the (<-) operator to assign values, or the (=) operator. The choice is mostly a matter of personal preference. I typically use (<-) for object assignment and (=) for parameter value assignment.

The sep (separator) parameter indicates how values on each line are separated. For example, (\t) would indicate tab-delimited values, and (" ") would indicate space-delimited values.

The print function displays the data table in memory. The print function has many optional parameters. Notice that the output in **Figure 1** displays data item indices starting at 1. For array, matrix and object indices, R is a 1-based language, rather than 0-based like the C# language.

The linear regression analysis is performed with these two R commands:

```
> model <- lm(data$Rate ~ (data$Color + data$Length + data$Width))
> summary(model)
```

You can interpret the first command as, "Store into an object named model the result of the lm (linear model) function analysis where the dependent variable to predict is the Rate column in the table object (data$Rate), and the independent predictor variables are Color, Length and Width." The second command means, "Display just the basic results of the analysis stored in the object named model."

The lm function generates a large amount of information. Suppose you wanted to predict the Rate value when the input values are Color = pink, Length = 5.0 and Width = 1.9. (Notice that this corresponds to data item [4], which has an actual Rate value of 0.4.) To make a prediction you'd use the values in the Estimate column:

```
Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)      -0.14758    0.48286  -0.306  0.77986
data$Colorpink -0.49083    0.04507 -10.891  0.00166 **
data$Colorteal  0.35672    0.09990   3.571  0.03754 *
data$Length     0.04159    0.07876   0.528  0.63406
data$Width      0.45200    0.11973   3.775  0.03255 *
```

If X represents the independent variable values, and if Y represents the predicted Rate, then:

```
X = (blue = NA, pink = 1, teal = 0, Length = 5.0, Width = 1.9)

Y = -0.14758 + (-0.49083)(1) + (0.35672)(0) + (0.04159)(5.0) + (0.45200)(1.9)
  = -0.14758 + (-0.49083) + (0) + (0.20795) + (0.85880)
  = 0.42834
```

Notice the predicted Rate, 0.43, is quite close to the actual rate, 0.40.

In words, to make a prediction using the model, you calculate a linear sum of products of the Estimate values multiplied by their corresponding X values. The Intercept value is a constant not associated with any variable. When you have categorical explanatory variables, one of the values is dropped (blue in this case).

The information at the bottom of the output display indicates how well the independent variables, Color, Length and Width, explain the dependent variable, Rate:

```
Residual standard error: 0.05179 on 3 degrees of freedom
Multiple R-squared:  0.9927,    Adjusted R-squared:  0.9829
F-statistic: 101.6 on 4 and 3 DF,  p-value: 0.00156
```

The multiple R-squared value (0.9927) is the percentage of variation in the dependent variable explained by the linear combination of the independent variables. Put slightly differently, R-squared is a value between 0 and 1 where higher values mean a better predictive model. Here, the R-squared value is extremely high, indicating Color, Length and Width can predict Rate very accurately. The F-statistic, adjusted R-squared value, and p-value are other measures of model fit.

One of the points of this example is that when programming using R, your biggest challenge by far is understanding the statistics behind the language functions. Most people learn R in an incremental way, by adding knowledge of one technique at a time, as needed to answer some specific question. A C# analogy would be learning about the various collection objects in the Collections.Generic namespace, such as the Hashtable and Queue classes. Most developers learn about one data structure at a time rather than trying to memorize information about all the classes before using any of them in a project.

Figure 2 **Contingency Matrix**

	Dem	Rep	Other	
Male	15	25	10	50
Female	30	15	5	50
	45	**40**	**15**	**100**

Another Chi-Square Test

The type of chi-square test in **Figure 1** is often called a test for uniform distribution because it tests if the observed data all have equal counts; that is, if the data is distributed uniformly. There are several other kinds of chi-square tests, including one called a chi-square test of independence.

Suppose you have a group of 100 people and you're interested if sex (male, female) is independent from political party affiliation (Democrat, Republican, Other). Imagine your data is in a contingency matrix as shown in **Figure 2**. It appears that perhaps males are more likely to be Republican than females are.

To use R to test if the two factors, sex and affiliation, are statistically independent, you'd first place the data into a numeric matrix with this command:

```
> cm <- matrix( c(15,30,25,15,10,5), nrow=2, ncol=3 )
```

Notice that in R, matrix data is stored by columns (top to bottom, left to right) rather than rows (left to right, top to bottom) as in C#.

The R chi-square test command is:

```
> chisq.test(cm)
```

The p-value result of the test is 0.01022, which indicates that at the 5 percent significance level, the two factors are not independent. In other words, there's a statistical relationship between sex and affiliation.

Notice in the first chi-square dice example, the input parameter is a vector, but in the second sex-affiliation example, the input is a matrix. The chisq.test function has a total of seven parameters, one required (a vector or a matrix), followed by six optional named parameters.

Like many C# methods in the Microsoft .NET Framework namespaces, most R functions are heavily overloaded. In C#, overloading is usually implemented using multiple methods with the same name but with different parameters. The use of generics is also a form of overloading. In R, overloading is implemented using a single function with many optional named parameters.

Graphs with R

As a C# programmer, when I want to make a graph of some program output data, I typically run my program, copy the output data, Ctrl+V paste that data into Notepad to remove weird control characters, copy that data, paste it into Excel, and then create a graph using Excel. This is kind of hacky, but it works fine in most situations.

One of the strengths of the R system is its native ability to generate graphs. Take a look at an example in **Figure 3**. This is a type of graph that isn't possible in Excel without an add-in of some sort.

In addition to the shell program shown in **Figure 1**, R also has a semi-GUI interface, RGui.exe, for use when you want to make graphs.

The graph in **Figure 3** shows the function z = f(x,y) = x * e^(-(x^2 + y^2)). The first three R commands to generate the graph are:

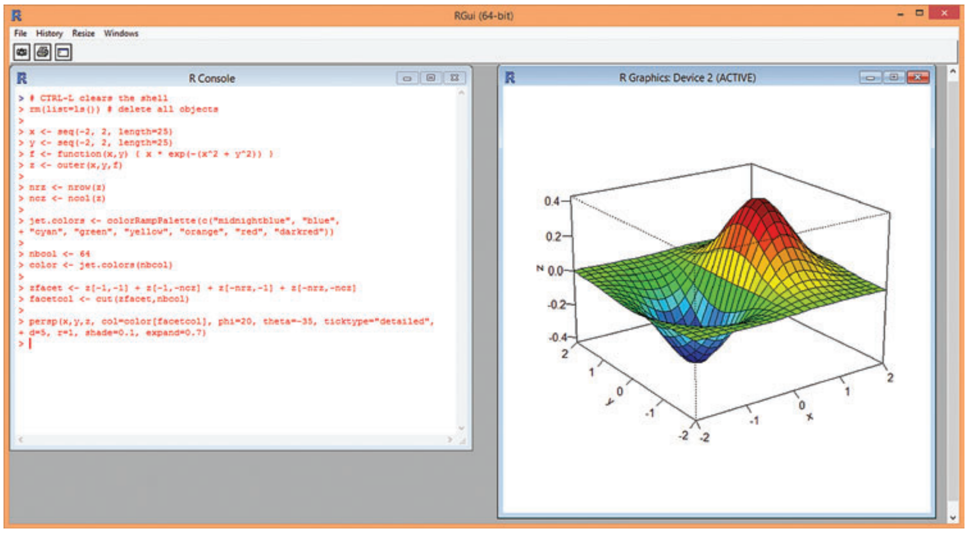
```
> rm(list=ls())
> x <- seq(-2, 2, length=25)
> y <- seq(-2, 2, length=25)
```

The rm function deletes an object from the current workspace in memory. The command used is a magic R incantation to delete all objects. The second and third commands create vectors of 25 values, evenly spaced, from -2 to +2 inclusive. I could've used the c function instead.

The next two commands are:

```
> f <- function(x,y) { x * exp(-(x^2
  + y^2)) }
> z <- outer(x,y,f)
```

The first command shows how to define a function in R using the function keyword. The built-in R function named outer creates a matrix of values using vectors x and y, and a function definition f. The result is a 25 x 25 matrix where the value in each cell is the value of function f that corresponds to x and y.



```
> # CTRL-L clears the shell
> rm(list=ls()) # delete all objects
>
> x <- seq(-2, 2, length=25)
> y <- seq(-2, 2, length=25)
> f <- function(x,y) { x * exp(-(x^2 + y^2)) }
> z <- outer(x,y,f)
>
> nrz <- nrow(z)
> ncz <- ncol(z)
>
> jet.colors <- colorRampPalette(c("midnightblue", "blue",
+ "cyan", "green", "yellow", "orange", "red", "darkred"))
>
> nbcol <- 64
> color <- jet.colors(nbcol)
>
> zfacet <- z[-1,-1] + z[-1,-ncz] + z[-nrz,-1] + z[-nrz,-ncz]
> facetcol <- cut(zfacet,nbcol)
>
> persp(x,y,z, col=color[facetcol], phi=20, theta=-35, ticktype="detailed",
+ d=5, r=1, shade=0.1, expand=0.7)
> 
```

Figure 3 **A 3D Graph Using R**

The next two commands are:

```
> nrz <- nrow(z)
> ncz <- ncol(z)
```

The nrow and ncol functions return the number of rows or the number of columns in their matrix argument. Here, both values would be 25.

The next R command uses the colorRampPalette function to create a custom color gradient palette to paint the graph:

```
> jet.colors <- colorRampPalette(c("midnightblue", "blue",
+ "cyan", "green", "yellow", "orange", "red", "darkred"))
```

When typing a long line in R, if you hit the <Enter> key, the system will jump the cursor down to the next line and place a + character as a prompt to indicate your command is not yet complete. Next:

```
> nbcol <- 64
> color <- jet.colors(nbcol)
```

The result of these two commands is a vector named color that holds 64 different color values ranging from a very dark blue, through green and yellow, to a very dark red. Next:

```
> zfacet <- z[-1,-1] + z[-1,-ncz] + z[-nrz,-1] + z[-nrz,-ncz]
> facetcol <- cut(zfacet,nbcol)
```

If you look closely at the graph in **Figure 3**, you'll see the surface is made of 25 x 25 = 625 small squares, or facets. The two preceding commands create a 25 x 25 matrix where the value in each cell is one of the 64 colors to use for the corresponding surface facet.

Finally, the 3D graph is displayed using the persp (perspective graph) function:

```
> persp(x,y,z, col=color[facetcol], phi=20, theta=-35,
+ ticktype="detailed", d=5, r=1, shade=0.1, expand=0.7)
```

The persp function has a lot of optional, named parameters. The col parameter is the color, or colors, to use. Parameters phi and theta set the viewing angle (left and right, and up and down) of the graph. Parameter ticktype controls how values on the x, y and z axes are displayed. Parameters r and d control the perceived eye distance to the graph, and the perceived 3D effect. The parameter named shade controls simulated shading from a virtual light source. The parameter named expand controls the ratio of the height and width of the graph. The persp function has many more parameters, but the ones used here will be sufficient in most situations.

This example points out that R has very powerful and flexible native graphing capabilities, but they tend to be relatively low level and require a fair amount of effort.

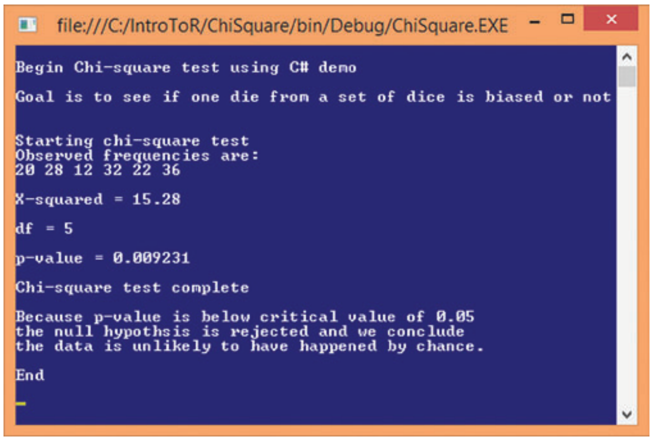
The Chi-Square Test in C#

To gain an understanding of the similarities and differences between R language analyses and C# language programming, it's useful to examine a C# implementation of a chi-square test. In addition, the C# code can make a nice addition to your personal code library. Take a look at the C# demo program in **Figure 4**.

The demo program approximates the R language chi-square dice test shown in **Figure 1**. Notice the output values of the C# demo are exactly the same as those in the R session.

To create the demo program, I launched Visual Studio and created a new C# console application project named ChiSquare. After the template code loaded into the editor, in the Solution Explorer window I right-clicked on file Program.cs and renamed it to ChiSquareProgram.cs and allowed Visual Studio to automatically rename class Program to ChiSquareProgram.

The complete C# demo program is listed in **Figure 5**. You'll notice that the source code is longer than you might expect. Implementing statistical programming using raw C# isn't overwhelmingly difficult, but the code does tend to be long.



```
Begin Chi-square test using C# demo

Goal is to see if one die from a set of dice is biased or not

Starting chi-square test
Observed frequencies are:
20 28 12 32 22 36

X-squared = 15.28

df = 5

p-value = 0.009231

Chi-square test complete

Because p-value is below critical value of 0.05
the null hypothesis is rejected and we conclude
the data is unlikely to have happened by chance.

End
```

Figure 4 **Chi-Square Test Using C#**

Figure 5 **C# Chi-Square Demo Program**

```
using System;
namespace ChiSquare
{
  class ChiSquareProgram
  {
    static void Main(string[] args)
    {
      try
      {
        Console.WriteLine("\nBegin Chi-square test using C# demo\n");
        Console.WriteLine(
          "Goal is to see if one die from a set of dice is biased or not\n");

        int[] observed = new int[] { 20, 28, 12, 32, 22, 36 };

        Console.WriteLine("\nStarting chi-square test");
        double p = ChiSquareTest(observed);
        Console.WriteLine("\nChi-square test complete");

        double crit = 0.05;
        if (p < crit)
        {
          Console.WriteLine("\nBecause p-value is below critical value of " +
            crit.ToString("F2"));
          Console.WriteLine("the null hypothsis is rejected and we conclude");
          Console.WriteLine("the data is unlikely to have happened by chance.");
        }
        else
        {
          Console.WriteLine("\nBecause p-value is not below critical value of " +
            crit.ToString("F2"));
          Console.WriteLine(
           "the null hypothsis is accepted (not rejected) and we conclude");
          Console.WriteLine("the observed data could have happened by chance.");
        }

        Console.WriteLine("\nEnd\n");
        Console.ReadLine();
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
      }

    } // Main

    static void ShowVector(int[] v)
    {
      for (int i = 0; i < v.Length; ++i)
        Console.Write(v[i] + " ");
      Console.WriteLine("");
    }

    static double ChiSquareTest(int[] observed)
    {
      Console.WriteLine("Observed frequencies are: ");
      ShowVector(observed);

      double x = ChiSquareStatistic(observed);
      Console.WriteLine("\nX-squared = " + x.ToString("F2"));

      int df = observed.Length - 1;
      Console.WriteLine("\ndf = " + df);

      double p = ChiSquareProb(x, df);
      Console.WriteLine("\np-value = " + p.ToString("F6"));

      return p;
    }

    static double ChiSquareStatistic(int[] observed)
    {
      double sumObs = 0.0;
      for (int i = 0; i < observed.Length; ++i)
        sumObs += observed[i];
      double expected = (int)(sumObs / observed.Length);

      double result = 0.0;
      for (int i = 0; i < observed.Length; ++i)
      {
        result += ((observed[i] - expected) *
         (observed[i] - expected)) / expected;
      }
      return result;
    }

    public static double ChiSquareProb(double x, int df)
    {
      // x = a computed chi-square value. df = degrees of freedom.
      // output = prob. the x value occurred by chance.
      // So, for example, if result < 0.05 there is only a 5% chance
      // that the x value occurred by chance and, therefore,
      // we conclude that the actual data which produced x is
      // NOT the same as the expected data.
      // This function can be used to create a ChiSquareTest procedure.
      // ACM Algorithm 299 and update ACM TOMS June 1985.
      // Uses custom Exp() function below.

      if (x <= 0.0 || df < 1)
        throw new Exception("parameter x must be positive " +
        "and parameter df must be 1 or greater in ChiSquaredProb()");

      double a = 0.0; // 299 variable names
      double y = 0.0;
      double s = 0.0;
      double z = 0.0;
      double e = 0.0;
      double c;
```

The Main method consists mostly of WriteLine statements. The essential calling code is:

```
int[] observed = new int[] { 20, 28, 12, 32, 22, 36 };
double p = ChiSquareTest(observed);
double crit = 0.05;
if (p < crit) {
  // Messages
} else {
  // Messages
}
```

The C# chi-square test accepts an array of observed values, computes the chi-square statistic value and displays it; computes and displays the degrees of freedom; and computes and returns the p-value.

Method ChiSquareTest calls three helper methods:

```
ShowVector(observed);
double x = ChiSquareStatistic(observed);
int df = observed.Length - 1;
double p = ChiSquareProb(x, df);
```

Method ShowVector displays the input vector, similar to the approach used by the R chisq.test function of echoing input parameter values. Method ChiSquareStatistic returns the calculated chi-square ("X-squared" in the R output), and method ChiSquareProb uses the return from ChiSquareStatistic to compute a probability (the "p-value" in the R output).

Method ChiSquareStatistic is a simple test for uniform distribution. The statistics equation for the chi-squared statistic is:

```
chi-squared = Sum( (observed - expected)^2 / expected )
```

So, before calculating chi-squared, you need to compute the expected values associated with the observed values. As explained earlier, to do this you add up the count value in the observed array (150 in the demo), then divide that sum by the number of values in the array (6 in the demo), to give the expected value (25) for all cells.

The most difficult part of writing a chi-square test is the calculation of the p-value. If you scan the definition of method ChiSquareProb in **Figure 5**, you'll quickly realize it requires deep, specialized knowledge. Furthermore, the method calls a helper method named Gauss that's equally complex.

Coding complicated numerical functions is actually quite easy because most functions have been solved for decades. Two of my

```
      bool even; // Is df even?

      a = 0.5 * x;
      if (df % 2 == 0) even = true; else even = false;

      if (df > 1) y = Exp(-a); // ACM update remark (4)

      if (even == true) s = y; else s = 2.0 * Gauss(-Math.Sqrt(x));

      if (df > 2)
      {
        x = 0.5 * (df - 1.0);
        if (even == true) z = 1.0; else z = 0.5;
        if (a > 40.0) // ACM remark (5)
        {
          if (even == true) e = 0.0;
          else e = 0.572364942924700087071713 5;
          c = Math.Log(a); // log base e
          while (z <= x)
          {
            e = Math.Log(z) + e;
            s = s + Exp(c * z - a - e); // ACM update remark (6)
            z = z + 1.0;
          }
          return s;
        } // a > 40.0
        else
        {
          if (even == true) e = 1.0;
          else e = 0.564189583547756286948079 5 / Math.Sqrt(a);
          c = 0.0;
          while (z <= x)
          {
            e = e * (a / z); // ACM update remark (7)
            c = c + e;
            z = z + 1.0;
          }
          return c * y + s;
        }
      } // df > 2
      else
      {
        return s;
      }
    } // ChiSquare()

    private static double Exp(double x) // ACM update remark (3)
    {
      if (x < -40.0) // ACM update remark (8)
        return 0.0;
      else
        return Math.Exp(x);
    }

    public static double Gauss(double z)
    {
      // input = z-value (-inf to +inf)
      // output = p under Normal curve from -inf to z
      // e.g., if z = 0.0, function returns 0.5000
      // ACM Algorithm #209
      double y; // 209 scratch variable
      double p; // result. called 'z' in 209
      double w; // 209 scratch variable

      if (z == 0.0)
        p = 0.0;
      else
      {
        y = Math.Abs(z) / 2;
        if (y >= 3.0)
        {
          p = 1.0;
        }
        else if (y < 1.0)
        {
          w = y * y;
          p = ((((((((0.000124818987 * w
            - 0.001075204047) * w + 0.005198775019) * w
            - 0.019198292004) * w + 0.059054035642) * w
            - 0.151968751364) * w + 0.319152932694) * w
            - 0.531923007300) * w + 0.797884560593) * y * 2.0;
        }
        else
        {
          y = y - 2.0;
          p = (((((((((((((-0.000045255659 * y
            + 0.000152529290) * y - 0.000019538132) * y
            - 0.000676904986) * y + 0.001390604284) * y
            - 0.000794620820) * y - 0.002034254874) * y
            + 0.006549791214) * y - 0.010557625006) * y
            + 0.011630447319) * y - 0.009279453341) * y
            + 0.005353579108) * y - 0.002141268741) * y
            + 0.000535310849) * y + 0.999936657524;
        }
      }

      if (z > 0.0)
        return (p + 1.0) / 2;
      else
        return (1.0 - p) / 2;
    } // Gauss()

  } // Program class

} // ns
```

most frequently used resources are the Association for Computing Machinery (ACM) Collected Algorithms repository, and the "Handbook of Mathematical Functions" by Abramowitz and Stegun (Dover Publications, 1965)—so well-known it's often just called "A&S." Both references are freely available in several places on the Web.

For example, method ChiSquareProb implements ACM Algorithm #299 and the helper method, Gauss, implements ACM Algorithm #209. An alternative to ACM #209 is A&S equation #7.1.26 plus a simple wrapping statement.

Many of the functions in the ACM Algorithms and A&S are implemented in existing C# libraries; however, most of those libraries are quite large. When possible, I prefer to code from scratch, using just the methods I need and avoiding external dependencies.

A Few Comments

This article shows only a tiny fraction of the R language but it should be enough to get you up and running. C# programmers tend to come into contact with R in two ways. First, you might use R to perform your own statistical analyses. As this article shows, using R is fairly easy, assuming you grasp the underlying statistics. Second, you might need to interact with someone who uses R as their primary language, and thus need to understand the R environment and terminology.

There are a few R language integrated development tools you can use. One well-known project is called RStudio. I generally prefer using the native R console. Several Microsoft products work with R. For example, the Microsoft Azure Machine Learning service can use R language scripts and I suspect support for R will be added to Visual Studio at some point. ■

Dr. James *McCaffrey works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. Dr. McCaffrey can be reached at jammc@microsoft.com.*

Thanks to the following Microsoft Research technical expert for reviewing this article: *Dan Liebling*

Linear Regression Using C#

The goal of a linear regression problem is to predict the value of a numeric variable based on the values of one or more numeric predictor variables. For example, you might want to predict the annual income of a person based on his education level, years of work experience and sex (male = 0, female = 1).

The variable to predict is usually called the dependent variable. The predictor variables are usually called the independent variables. When there's just a single predictor variable, the technique is sometimes called simple linear regression. When there are two or more predictor variables, the technique is generally called multiple, or multivariate, linear regression.

A good way to see where this article is headed is to take a look at the demo program in **Figure 1**. The C# demo program predicts annual income based on education, work and sex. The demo begins by generating 10 synthetic data items. Education level is a value between 12 and 16. Work experience is a value between 10 and 30. Sex is an indicator variable where male is the reference value, coded as 0, and female is coded as 1. Income, in thousands of dollars, is in the last column. In a non-demo scenario, you'd probably read data from a text file using a method named something like MatrixLoad.

R-squared is a value between 0 and 1 that describes how well the prediction model fits the raw data.

After generating the synthetic data, the demo program uses the data to create what's called a design matrix. A design matrix is just the data matrix with a leading column of all 1.0 values added. There are several different algorithms that can be used for linear regression; some can use the raw data matrix while others use a design matrix. The demo uses a technique that requires a design matrix.

After creating the design matrix, the demo program finds the values for four coefficients, (12.0157, 1.0180, 0.5489, -2.9566). The coefficients are sometimes called b-values or beta-values. The first value, 12.0157, is usually called the intercept. It's a constant not associated with any predictor variable. The second, third and fourth coefficient values (1.0180, 0.5489, -2.9566) are associated with education level, work experience and sex, respectively.

The very last part of the output in **Figure 1** uses the values of the coefficients to predict the income for a hypothetical person who has an education level of 14; 12 years of work experience; and whose sex is 0 (male). The predicted income is 32.86, which is calculated like so:

$$\begin{aligned}\text{income} &= 12.0157 + (1.0180)(14) + (0.5489)(12) + (-2.9566)(0)\\ &= 12.0157 + 14.2520 + 6.5868 + 0\\ &= 32.86\end{aligned}$$

In other words, to make a prediction using linear regression, the predictor values are multiplied by their corresponding coefficient values and summed. It's very simple. Notice that the leading intercept value (12.0157 in the example) can be considered a coefficient associated with a predictor variable that always has a value of 1. This fact, in part, explains the column of 1.0 values in the design matrix.

The essence of a linear regression problem is calculating the values of the coefficients using the raw data or, equivalently, the design matrix. This is not so easy. The demo uses a technique called closed form matrix inversion, also known as the ordinary least squares method. Alternative techniques for finding the values of the coefficients include iteratively reweighted least squares, maximum likelihood estimation, ridge regression, gradient descent and several others.

In **Figure 1**, before the prediction is made, the demo program computes a metric called the R-squared value, which is also called the coefficient of determination. R-squared is a value between 0 and 1 that describes how well the prediction model fits the raw data. This is sometimes expressed as, "the percentage of variation explained by the model." Loosely interpreted, the closer R-squared is to 1, the better the prediction model is. The demo value of 0.7207, or 72 percent, would be considered relatively high (good) for real-world data.

This article assumes you have at least intermediate C# programming skills, but doesn't assume you know anything about linear regression. The demo program is too long to present in its entirety, but the complete source code is available in the download that accompanies this article.

Code download available at msdn.microsoft.com/magazine/msdnmag0715.

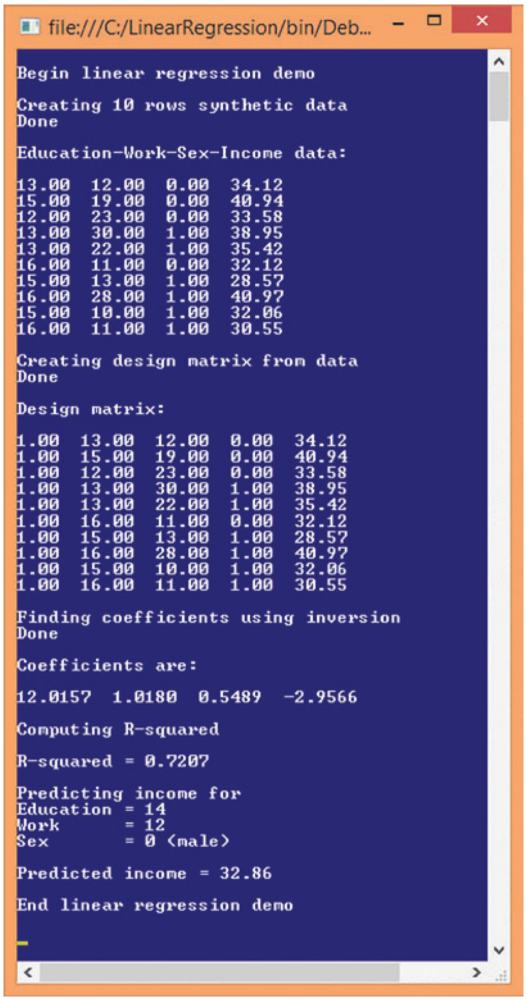
Understanding Linear Regression

Linear regression is usually best explained using a diagram. Take a look at the graph in **Figure 2**. The data in the graph represents predicting annual income from just a single variable, years of work experience. Each of the red dots corresponds to a data point. For example, the leftmost data item has work = 10 and income = 32.06. Linear regression finds two coefficients: one intercept and one for the work variable. As it turns out, the values of the coefficients are 27.00 and 0.43.

The coefficient values determine the equation of a line, which is shown in blue in **Figure 2**. The line (the coefficients) minimizes the sum of the squared deviations between the actual data points (yi) and the predicted data points (fi). Two of the 10 deviations are shown with dashed lines in **Figure 2**. The first deviation shown is yi - fi = 28.6 - 32.6 = -4.0. Notice that deviations can be positive or negative. If the deviations weren't squared, the negatives and positives could cancel each other out.

The graph in **Figure 2** shows how simple linear regression, with just one independent variable, works. Multivariate linear regression extends the same idea—find coefficients that minimize the sum of squared deviations—using several independent variables.

Expressed intuitively, linear regression finds the best line through a set of data points. This best line can be used for prediction. For example, in **Figure 2**, if a hypothetical person had 25 years of work experience, her predicted income on the blue line would be about 38.



```
Begin linear regression demo

Creating 10 rows synthetic data
Done

Education-Work-Sex-Income data:

13.00  12.00  0.00  34.12
15.00  19.00  0.00  40.94
12.00  23.00  0.00  33.58
13.00  30.00  1.00  38.95
13.00  22.00  1.00  35.42
16.00  11.00  0.00  32.12
15.00  13.00  1.00  28.57
16.00  28.00  1.00  40.97
15.00  10.00  1.00  32.06
16.00  11.00  1.00  30.55

Creating design matrix from data
Done

Design matrix:

1.00  13.00  12.00  0.00  34.12
1.00  15.00  19.00  0.00  40.94
1.00  12.00  23.00  0.00  33.58
1.00  13.00  30.00  1.00  38.95
1.00  13.00  22.00  1.00  35.42
1.00  16.00  11.00  0.00  32.12
1.00  15.00  13.00  1.00  28.57
1.00  16.00  28.00  1.00  40.97
1.00  15.00  10.00  1.00  32.06
1.00  16.00  11.00  1.00  30.55

Finding coefficients using inversion
Done

Coefficients are:

12.0157  1.0180  0.5489  -2.9566

Computing R-squared

R-squared = 0.7207

Predicting income for
Education = 14
Work      = 12
Sex       = 0 (male)

Predicted income = 32.86

End linear regression demo
```

Figure 1 **Linear Regression Using C#**

Solving the Least Squares Equation

If a linear regression problem has n predictor variables, then n+1 coefficient values must be found, one for each predictor plus the intercept value. The demo program uses the most basic technique to find the coefficient values. The values of the coefficients are often given using the somewhat intimidating equation shown in **Figure 3**. The equation is not as complicated as it might first appear.

The Greek letter beta resembles a script B and represents the coefficient values. Notice that all the letters in the equation are in bold, which in mathematics indicates they represent multi-valued objects (matrices or arrays/vectors) rather than simple scalar values (plain numbers). Uppercase X represents the design matrix. Uppercase X with a T exponent means the transpose of the design matrix. The * symbol means matrix multiplication. The -1 exponent means matrix inversion. Uppercase Y is a column vector (a matrix with one column) of the dependent variable values. Therefore, solving for the values of the coefficients really means understanding matrix operations.

The diagrams in **Figure 4** illustrate matrix transposition, matrix multiplication, and matrix inversion. The transpose of a matrix just swaps rows and columns. For example, suppose you have a 2x3 matrix, that is, one with 2 rows and 3 columns. The transpose of the matrix will be 3x2 where the rows of the original matrix become the columns of the transpose matrix.

Matrix multiplication may seem a bit odd if you haven't encountered it before. If you multiply an (n x m) sized matrix times an (m x p) sized matrix, the result is an (n x p) sized matrix. For example, a 3x4 * a 4x2 matrix has size 3x2. A detailed discussion of matrix multiplication is outside the scope of this article, but once you've seen a few examples, the process is easy to understand and easy to implement in code.

The third matrix operation needed to solve for linear regression coefficient values is matrix inversion, which, unfortunately, is difficult to grasp and difficult to implement. For the purposes of this article, it's enough to know that the inverse of a matrix is defined only when the matrix has the same number of rows and columns (a square matrix). **Figure 4** shows a 3x3 matrix and its inverse.

There are several algorithms that can be used to find the inverse of a matrix. The demo program uses a technique called Doolittle's decomposition.

To summarize, a linear regression problem with n predictor variables involves finding the values for n+1 coefficients. This can be done using matrices with matrix transposition, matrix multiplication and matrix inversion. Transposition and multiplication are easy, but finding the inverse of a matrix is difficult.

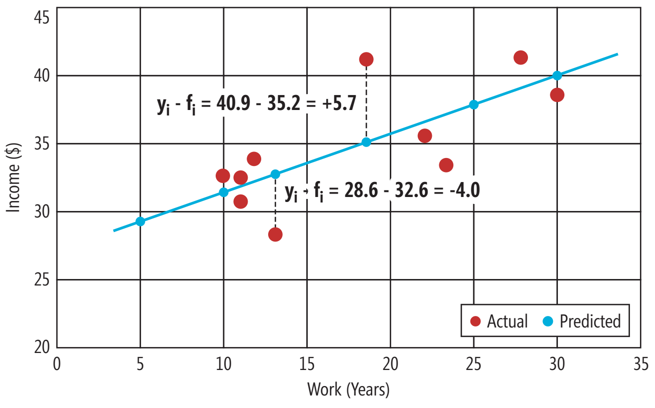


Figure 2 **Linear Regression with One Independent Variable**

The Demo Program Structure

To create the demo program, I launched Visual Studio and selected the console application project template. I named the project LinearRegression. The program has no significant .NET Framework dependencies so any version of Visual Studio will work.

After the template code loaded into the editor, in the Solution Explorer window, I right-clicked on file Program.cs and renamed it to LinearRegressionProgram.cs. I allowed Visual Studio to automatically rename class Program. At the top of the Editor window, I deleted all using statements except the one referencing the top-level System namespace.

The overall structure of the demo program, with a few minor edits to save space, is presented in **Figure 5**. All the program control logic is in the Main method. The demo program uses a static-method approach rather than an OOP approach.

$$\beta = (X^T * X)^{-1} * X^T * Y$$

Figure 3 **Linear Regression Coefficients Solution Using Matrices**

The demo program defines a matrix in the simplest way possible, as an array of arrays.

Method Income returns a predicted income from input parameters with values for education level, work experience and sex, using an array of coefficient values. Method RSquared returns the R-squared value of the model from the data and the coefficients. Method DummyData generates the synthetic data used for the demo.

Method Design accepts a matrix of data and returns an augmented design matrix with a leading column of 1.0 values. Method Solve accepts a design matrix and uses matrix operations to find the linear regression coefficients.

Most of the hard work is done by a set of static methods that perform matrix operations. The demo program defines a matrix in the simplest way possible, as an array of arrays. An alternative is to create a program-defined Matrix class, but in my opinion that approach is unnecessarily complicated. Sometimes ordinary arrays are better than program-defined objects.

Method MatrixTranspose returns the transposition of a matrix. Method MatrixProduct returns the result of multiplying two matrices. Method MatrixInverse returns the inverse of a matrix. The demo has many helper methods. In particular, method MatrixInverse calls helper methods MatrixDuplicate, MatrixDecompose and HelperSolve.

The Solve Method

The heart of the linear regression demo program is method Solve. The method's definition begins with:

```
static double[] Solve(double[][] design)
{
  int rows = design.Length;
  int cols = data[0].Length;
  double[][] X = MatrixCreate(rows, cols - 1);
  double[][] Y = MatrixCreate(rows, 1);
...
```

The single input parameter is a design matrix. An alternative approach you might want to consider is to pass the source data matrix and then have Solve call helper method Design to get the design matrix. Helper method MatrixCreate allocates space for, and returns, a matrix with the specified number of rows and columns. The local X matrix holds the values of the independent variables (with a leading 1.0 value). The local Y matrix has just one column and holds the values of the dependent variable (annual income in the demo).

Next, the cells in matrices X and Y are populated using the values in the design matrix:

```
int j;
for (int i = 0; i < rows; ++i)
{
  for (j = 0; j < cols - 1; ++j)
  {
    X[i][j] = design[i][j];
  }
  Y[i][0] = design[i][j]; // Last column
}
```

Notice that index variable j is declared outside the nested for loops so it can be used to populate the Y matrix. With the X and Y matrices in hand, the linear regression coefficients can be found according to the equation shown in **Figure 3**:

```
...
  double[][] Xt = MatrixTranspose(X);
  double[][] XtX = MatrixProduct(Xt, X);
  double[][] inv = MatrixInverse(XtX);
  double[][] invXt = MatrixProduct(inv, Xt);
  double[][] mResult = MatrixProduct(invXt, Y);
  double[] result = MatrixToVector(mResult);
  return result;
} // Solve
```

In the demo, matrix X has size 10x4 so its transpose, Xt, has size 4x10. The product of Xt and X has size 4x4 and the inverse, inv, also has size 4x4. In general, for a linear regression problem with n independent predictor variables, when using the matrix inversion technique you'll need to find the inverse of a matrix with size (n+1) x (n+1). This means the inversion technique isn't suitable for linear regression problems that have a huge number of predictor variables.

The product of the 4x4 inverse matrix and the 4x10 transpose matrix, invXt in the code, has size 4x10. The product of

$$\underset{\text{M is (2x3)}}{\begin{bmatrix}1.0&2.0&3.0\\4.0&5.0&6.0\end{bmatrix}} \rightarrow \underset{\text{M}^{T}\text{ (transpose) is (3x2)}}{\begin{bmatrix}1.0&4.0\\2.0&5.0\\3.0&6.0\end{bmatrix}} \qquad \underset{\text{M is (3x3)}}{\begin{bmatrix}1.0&2.0&3.0\\4.0&6.0&1.0\\5.0&7.0&4.0\end{bmatrix}} \rightarrow \underset{\text{M}^{-1}\text{ (inverse) is (3x3)}}{\begin{bmatrix}-0.6441&0.2203&0.1525\\0.2373&-0.1864&0.1017\\-.3898&0.0508&-0.1186\end{bmatrix}}$$

$$\underset{\text{A is (3x4)}}{\begin{bmatrix}1.0&2.0&3.0&4.0\\5.0&6.0&7.0&8.0\\9.0&0.0&1.0&2.0\end{bmatrix}} * \underset{\text{B is (4x2)}}{\begin{bmatrix}1.0&2.0\\3.0&4.0\\5.0&6.0\\7.0&8.0\end{bmatrix}} \rightarrow \underset{\text{A * B (product) is (3x2)}}{\begin{bmatrix}50.0&60.0\\114.0&140.0\\28.0&40.0\end{bmatrix}}$$

Figure 4 **Three Matrix Operations Used to Find Linear Regression Coefficients**

invXt and the 10x1 Y matrix, mResult ("matrix result") in the code, has size 4x1. These values are the coefficients you need. For convenience, the values in the single column Y matrix are transferred to an ordinary array using helper method MatrixToVector.

Calculating R-Squared

As noted earlier, the R-squared metric is a measure of how well the actual data points fit the computed regression line. In math terms, R-squared is defined as R2 = 1 - (SSres / SStot). The SSres term is usually called the "residual sum of squares." It's the sum of the squared differences between the actual Y values and the predicted Y values, as illustrated in the graph in **Figure 2**. The SStot term is the "total sum of squares." It's the sum of the squared differences between each actual Y value and the mean (average) of all of the actual Y values.

The R-squared metric in linear regression is also called the coefficient of determination and is related to, but different from, another statistical metric named r-squared ("little r-squared"). Interpreting R-squared is a bit tricky and depends on the particular problem domain under investigation. For the natural and social sciences, where data is typically messy and incomplete, an R-squared value of 0.6 or greater is often considered pretty good.

Figure 5 **Linear Regression Demo Program Structure**

```
using System;
namespace LinearRegression
{
  class LinearRegressionProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin linear regression demo");

      // Generate synthetic data
      // Create design matrix
      // Solve for LR coefficients
      // Calculate R-squared value
      // Do a prediction

      Console.WriteLine("End linear regression demo");
      Console.ReadLine();
    }

    static double Income(double x1, double x2,
      double x3, double[] coef) { . . }

    static double RSquared(double[][] data,
      double[] coef) { . . }

    static double[][] DummyData(int rows,
      int seed) { . . }

    static double[][] Design(double[][] data) { . . }

    static double[] Solve(double[][] design) { . . }

    static void ShowMatrix(double[][] m, int dec) { . . }

    static void ShowVector(double[] v, int dec) { . . }

    // ----------

    static double[][] MatrixTranspose(double[][] matrix)
      { . . }

    static double[][] MatrixProduct(double[][] matrixA,
      double[][] matrixB) { . . }

    static double[][] MatrixInverse(double[][] matrix)
      { . . }

    // Other matrix routines here
  }
} // ns
```

As noted earlier, the R-squared metric is a measure of how well the actual data points fit the computed regression line.

There's an alternative measure of the variance explained by the regression model called the adjusted R-squared. This metric takes into account the number of predictor variables and the number of data items. For most purposes, using the plain R-squared value is good enough to get an idea of the predictive quality of a linear regression model.

Wrapping Up

If you search the Internet for examples of how to perform linear regression using a programming language, you won't find very many references. I think there are two main reasons for this relative lack of information. First, solving for linear regression coefficients using matrix operations is quite difficult, mostly because of the matrix inversion operation. In some ways, I consider the MatrixInverse method of the demo program to be among the most complicated code routines I've ever written. Second, there are plenty of existing standalone tools that can perform linear regression, in particular, the Excel spreadsheet program with its Data Analysis add-in. It's relatively rare to need to directly embed linear regression solution code in a software system.

Linear regression has been studied for decades and there are many ways to extend the technique. For example, you can introduce what are called interaction effects that combine two or more predictor variables. These extensions are sometimes called general linear models to distinguish them from the basic form of linear regression.

In my opinion, linear regression is the "Hello World" technique of classical statistics. There's no clear, universally agreed upon distinction between classical statistics and machine learning, but I tend to think of classical statistics techniques as those that were first studied by mathematicians starting in the early 1900s. In my mind, machine learning techniques, like neural network classification, are those that are more recent, first appearing in the 1950s. Classical statistics linear regression is closely related to a machine learning technique called logistic regression, which has been the topic of several Test Run columns. ■

Dr. James McCaffrey *works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. Dr. McCaffrey can be reached at jammc@microsoft.com.*

Thanks to the following Microsoft Research technical expert for reviewing this article: *Charles Parker*

100 Years of Solitaire

OK, it's not exactly 100 years, but 25 years ago this month Microsoft took the watershed step toward becoming the colossus it is today. I speak, of course, of releasing Solitaire with Windows 3.0.

You cannot discuss the social history of the last quarter-century in the computing industry—roughly half its entire lifetime—without acknowledging the impact of Solitaire. As I wrote in "Understanding COM+" (Microsoft Press, 1999), "Windows versions 1 and 2 flailed around irrelevantly .... Microsoft practically gave them away in Cracker Jack boxes, and still no one would use them .... Windows 3 produced a pretty good MS-DOS multitasker that also played Solitaire, and the rest is history."

The DOS multitasker attracted the serious geeks; probably including you, dear reader. But it was Solitaire that brought ordinary users into Windows. It was so much easier to use than the DOS apps on the same 386 box. You selected commands from menus, and dragged cards with the mouse, instead of typing in obscure alphabetic sequences. And the graphical display, actually seeing a pretty picture, instead of text scrolling by too fast to read—oh, joy.

Solitaire demonstrated that these hitherto-boring boxes could inject some fun into your life. As Josh Levin wrote in the online magazine *Slate*: "Moving a black two onto a red three may not have seemed particularly enticing on its own terms, but compared with the visual stimuli provided by an Excel spreadsheet, a post-victory card cascade was an unimaginably rousing spectacle."

And once you had tried it, you couldn't ignore it. Waiting for a phone call, or just didn't feel like tackling that pile of work? No problem. "Just one quick game. Dang, lost that one, another to break even? Great, I won. A rubber game? Why not? Aw, heck, the obstetrician can leave a voice mail." (That might have been the first example of the irresistible force that draws your hand to your mobile phone, of which I wrote in February 2012: msdn.microsoft.com/magazine/hh781031.)

It could get out of hand. I remember teaching a Windows programming class (16-bit SDK in C) and getting annoyed at students playing Solitaire instead of listening to me. So I booby-trapped the lab computers, redirecting the Program Manager Solitaire icon to activate a noisemaker program. When one student's PC started blaring, I turned on her, shouting, "A-HA! I caught you playing SOLITAIRE!" She blushed brick red, and wanted to fall through the floor. No one dared touch Solitaire for the rest of the class. (Joanna, if you're reading this, I'm sorry. Sort of.)

Solitaire evolved and grew along with Windows. I remember the 1992 PDC in San Francisco, introducing Windows NT to developers. The GDI team explained how kernel-user mode transitions had degraded the victory cascade in Solitaire. They wisely announced the workaround at the same time, for surely that crowd would have defenestrated anyone who violated their beloved Solitaire.

We marked Windows milestones by their Solitaire variants. Windows NT gave us FreeCell, which we loved because almost every game was winnable. Windows XP gave us Spider, where you could cheat by uncovering cards, then using Undo. Vista didn't have a new Solitaire game. Do you wonder that it flopped?

The DOS multitasker attracted the serious geeks; probably including you, dear reader. But it was Solitaire that brought ordinary users into Windows.

Solitaire isn't built into Windows 8, though you can get it free in the Windows Store. Perhaps Microsoft figured that Solitaire would draw users into the store, as it drew them to the original desktop. But it's one thing to give users something new and great; an entirely different thing to remove something they love and make them work to get it back again. The firestorm over removing the Start menu was nothing compared to the *pool-pah* over removing Solitaire. No wonder that SKU tanked. Microsoft had to backtrack in Windows 10, restoring the Start menu due to popular demand. I hope the company comes to its senses regarding Solitaire, as well.

This column was fun to write. But now I have to wipe those Solitaire programs off my machine, because I'm not getting a damn thing done. I tell myself it's research. Oh, heck, just one more game. Satya can leave a voice mail. ■

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.*

WPF lives!

XCEED Business Suite for WPF

The essential set of WPF controls for all your line-of-business solutions. Includes the industry-leading Xceed DataGrid for WPF. A total of 85 tools!

XCEED
MULTI-TALENTED COMPONENTS
xceed.com

650 CONTROLS

•••

12 PLATFORMS

•••

A $9,975 VALUE

ESSENTIAL STUDIO ENTERPRISE EDITION

COMMUNITY LICENSE

ABSOLUTELY FREE

WHAT DO YOU GET?

- ✓ A license that never expires.
- ✓ Free access to our entire product offering.
- ✓ A license to build commercial applications.

WHO QUALIFIES?

Individual developers, or up to five users at companies with less than $1 million USD in annual gross revenue, are eligible for the Community License.

www.syncfusion.com/communitylicense

Syncfusion®