

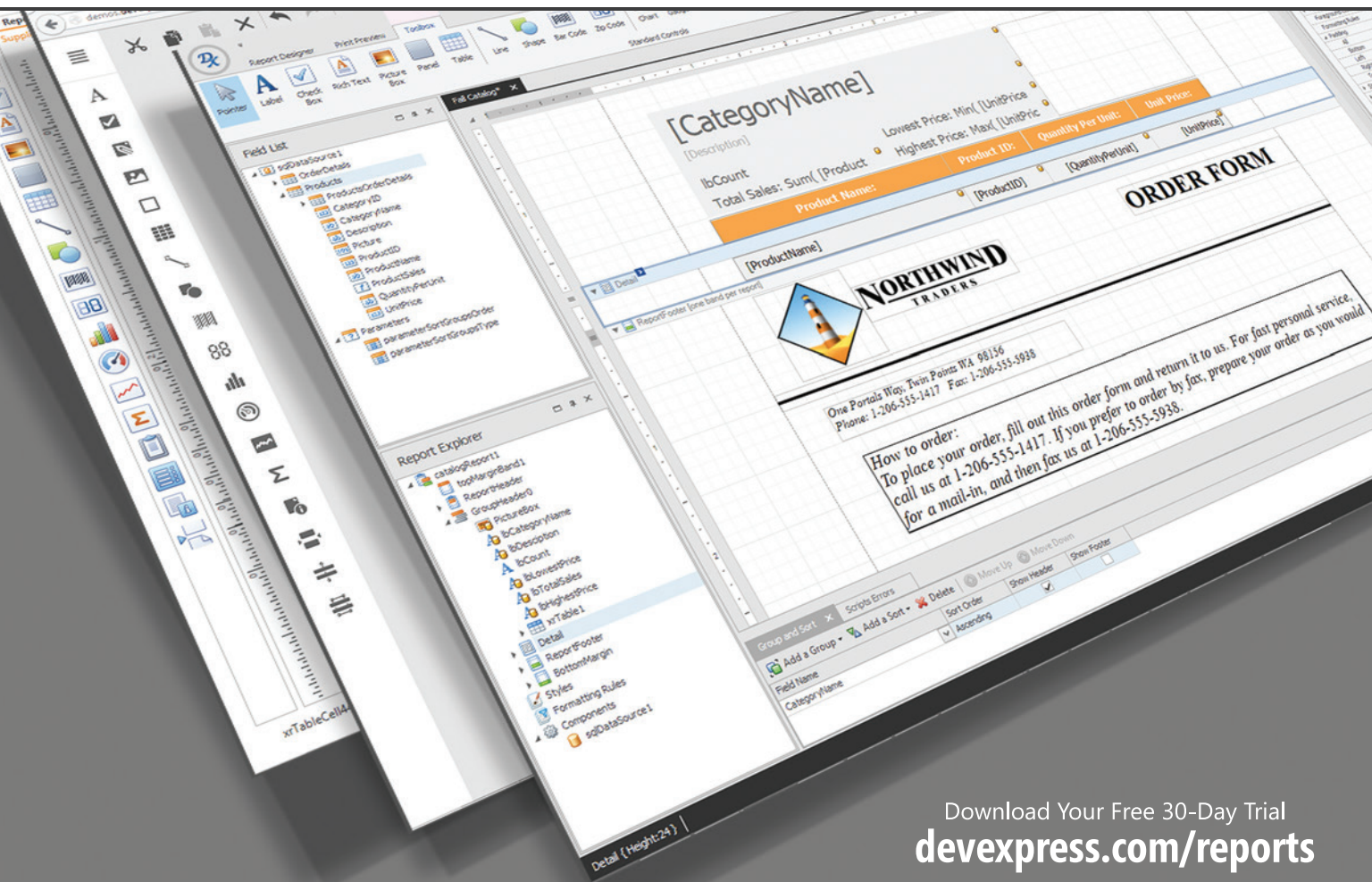
msdn magazine



Build Cloud-Connected
Xamarin Apps.....28

The Easier Way to Create Reports

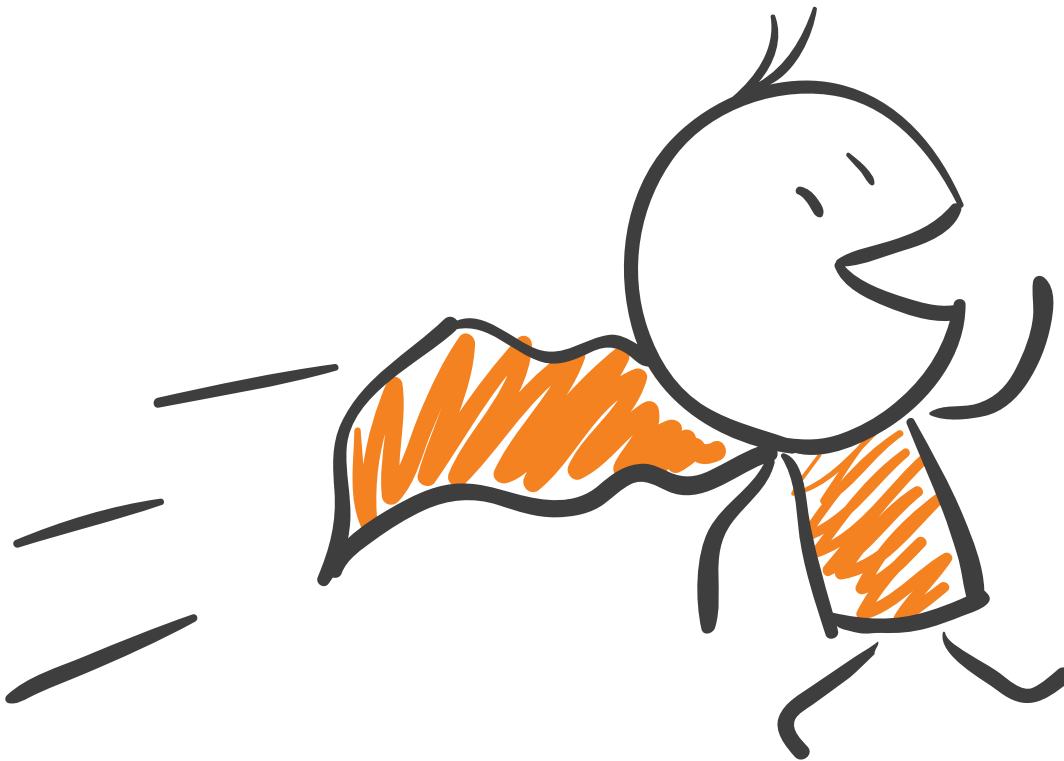
A Report Platform optimized for
WPF, ASP.NET and Windows Forms developers.



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

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

msdn magazine



**Build Cloud-Connected
Xamarin Apps.....28**

Build a Xamarin App with Authentication and Offline Support Kraig Brockschmidt, Mike Wasson, Rick Anderson, Tom Dykstra and Erik Reitan	28
Fault Tolerance Pitfalls and Resolutions in the Cloud Mario Szpuszta and Srikumar Vaitinadin	38
Creating Unified, Heroku-Style Workflows Across Cloud Platforms Bruno Terkaly	48
Streamline Code with Native Profile-Guided Optimization Hadi Brais	52
Protect Sensitive Information with Azure Key Vault Rahul Nath	58
Enabling DevOps on the Microsoft Stack Micheal Learned	64

COLUMNS

UPSTART

Radio Free Internet
Keith Boyd, page 6

CUTTING EDGE

Event Sourcing for the
Common Application
Dino Esposito, page 10

WINDOWS WITH C++

Classy Types in the
Windows Runtime
Kenny Kerr, page 14

DATA POINTS

Revisiting JavaScript Data
Binding—Now with Aurelia
Julie Lerman, page 18

TEST RUN

Computing with Artificial
Spiking Neurons
James McCaffrey, page 70

THE WORKING PROGRAMMER

How To Be MEAN: Node.js
Ted Neward, page 74

MODERN APPS

Usability Practices
for Modern Apps
Rachel Appel, page 76

DON'T GET ME STARTED

Darwin's Camera
David Platt, page 80

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



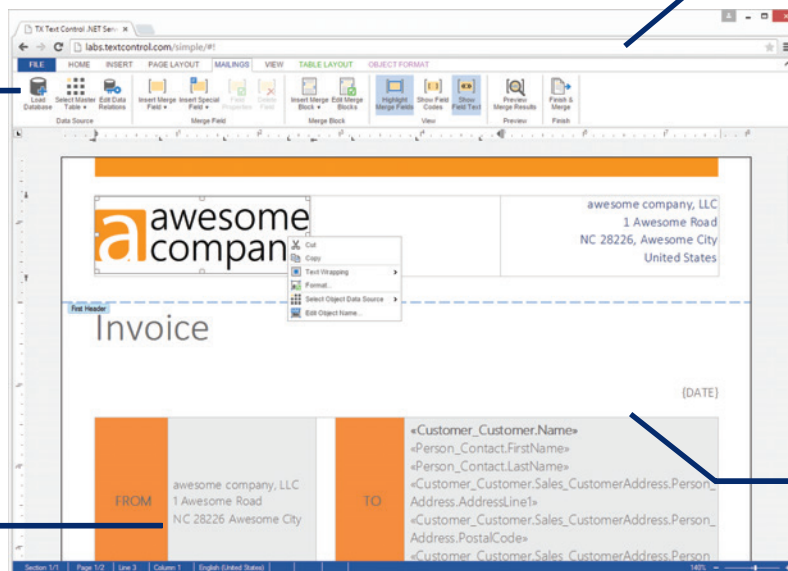
Database support for ADO.NET, ODBC, DataSet, DataTable and all IEnumerable business objects



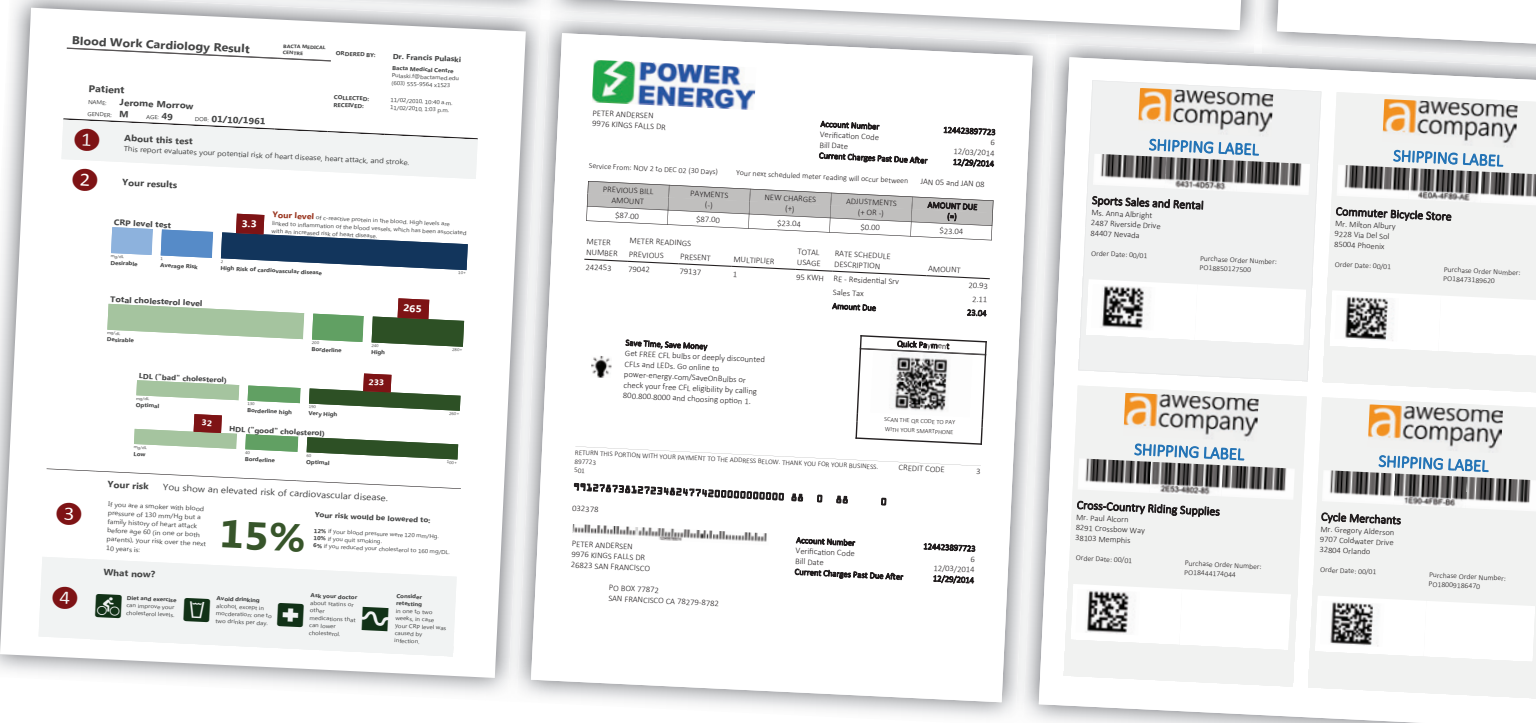
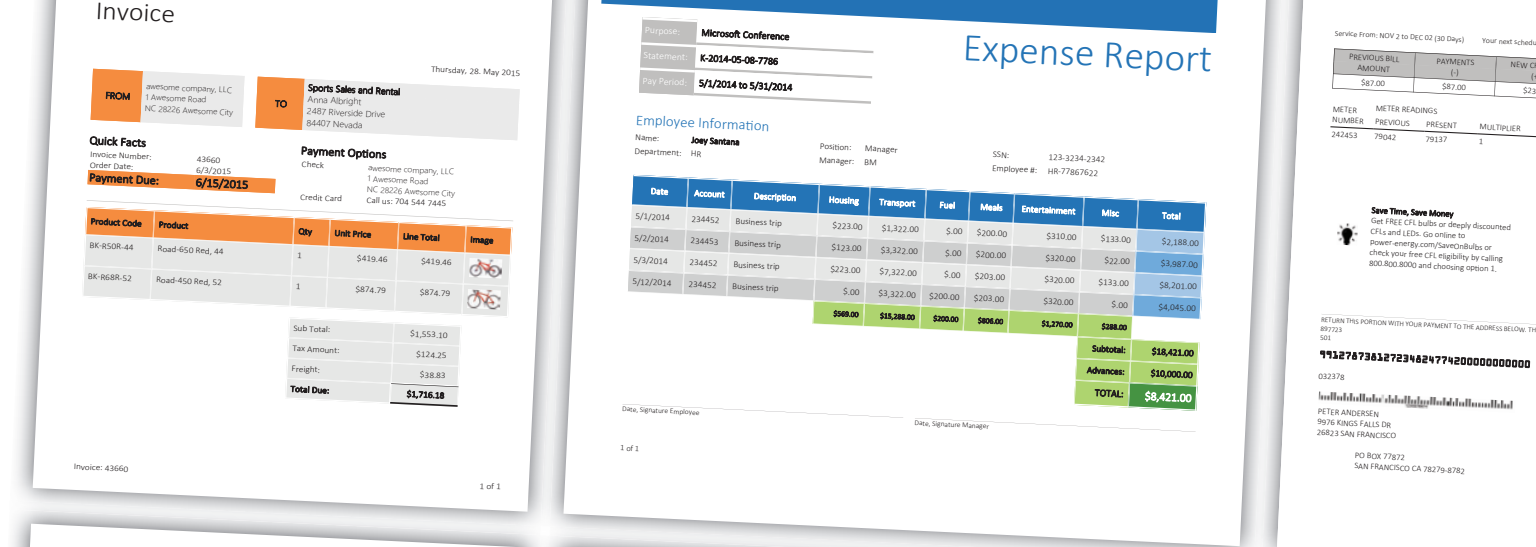
Cross-browser, cross-platform document and template editing



Create Adobe PDF and PDF/A documents



MS Word compatible templates and MS Word inspired UI



Migrating from Office Automation or another reporting tool? Get your free consultation today.

Call: +1 855-533-TEXT

Live demos and 30-day trial version download at:
www.textcontrol.com/reporting

X12

TEXTCONTROL

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



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

Director, Site Administration Shane Lee

Site Administrator Biswarup Bhattacharjee

Front-End Developer Anya Smolinski

Junior Front-End Developer Casey Rysavy

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

Marketing Coordinator Alicia Chew

Marketing & Editorial Assistant Dana Friedman

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



YOUR GROWTH. OUR BUSINESS.

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: jloug@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 contents herein do not necessarily express those of the publisher.





MILLIONS OF LINES OF CODE AT YOUR FINGERTIPS

Document Imaging

Document Viewer

Document Converter

OCR, MICR, OMR & ICR

Barcode & Forms Recognition

Create, Save, Edit & View PDF

Annotations, TWAIN & SVG

Medical Imaging

DICOM, CCOW & HL7

PACS Client & Server Framework

DICOM Storage Server Framework

Web & Desktop Viewers

Image Processing & Annotations

Medical 3D (MPR, MIP, VRT)

Multimedia Imaging

Play, Capture, Convert & DVR

Stream - MPEG2-TS & RTSP

Media Streaming Server

MPEG-4, H.264 & H.265

Media Foundation & DirectShow

Distributed Transcoding

.NET Windows API WinRT Linux iOS OS X Android HTML5





The Internet of Car Wrecks

I used to drive a Jeep Wrangler and I loved the bare bones character of the little truck. Hand-cranked windows, manual locks, a five-speed manual transmission, and doors that came off with just the turn of a couple nuts. There's something to be said for luddite simplicity and ruggedness—especially given mounting evidence that new automobiles may be vulnerable to attack via their Internet-connected infotainment systems.

It begs the question: In what world did it seem like a good idea for mission-critical controls to share trusted network space with infotainment systems linked to the public Internet?

A recent article from *Wired* magazine (wrd.cm/1KvuymJ) describes how security experts Charlie Miller and Chris Valasek remotely hijacked a Jeep Cherokee SUV, using a zero-day exploit to hack into the car's UConnect system, which operates on the Sprint cellular network and provides entertainment, navigation and communication functionality. From there, the article reports, they were able to rewrite firmware in the car's entertainment head unit, which won them access to the controller area network (CAN) bus. The CAN bus is an industry standard conduit for car-based microcontrollers and devices to communicate with each other. Once Miller and

Valasek had access to that, they gained a level of control over steering, brakes, throttle, and a host of other critical functions and controls.

It begs the question: In what world did it seem like a good idea for mission-critical controls to share trusted network space with infotainment systems linked to the public Internet? The CAN bus is inherently insecure. It's unencrypted, provides no segmentation or firewalling of components on the network, and has no ability to discern malicious commands from legitimate ones.

In the article, author Andy Greenberg recounts driving as Miller and Valasek—working on a laptop 10 miles away—went to work, turning on the air-conditioning, blaring the radio at full volume, and activating the windshield wipers as wiper fluid sprayed across the glass. Inside the car, Greenberg was unable to counter any of these actions. A few minutes later Miller and Valasek disabled the transmission, causing the car to slow in traffic even as Greenberg revved the engine. In a later demo, the two disabled the car's brakes, causing the vehicle to settle into a ditch.

This isn't the first such demonstration. "60 Minutes" in February broadcast a report (cbsn.ws/1VMbHHI) that showed a sedan driven by CBS correspondent Leslie Stahl being wirelessly hijacked by Dan Kaufman, head of the Information Innovation Office at the Defense Advanced Research Projects Agency (DARPA).

I see all this as an early test of the Internet of Things (IoT) concept. Carmakers are not uniquely negligent in securing IoT systems—by most accounts, almost everyone is bad at it—but the risk they face is severe. As Miller and Valasek showed, a hijacked car could result in injury or death.

Carmakers are getting the message, as evidenced by Fiat Chrysler's recall of 1.4 million UConnect-equipped cars just two days after *Wired* published its story. I just hope device and system makers in other industries are paying full attention.

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.

We didn't invent the Internet...

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



TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

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



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

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

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

connectivity
powered by 

To learn more please visit our website →

www.nsoftware.com

Radio Free Internet

The year was 1994. Ace of Bass, Boyz II Men and Celine Dion ruled the airwaves—at least on commercial radio. But up in the college town of Bellingham, Wash. (population 50,000), a plucky little radio station called KUGS FM was playing less conventional fare. From Hawaiian music to Celtic, from Funk to Metal, KUGS showcased an eclectic mix of styles, anchored by straight-up college rock and roll. The only problem? A 100-watt transmitter that lacked power to reach the entirety of the college campus, much less the surrounding area. That tiny transmitter kept listenership low, but it did little to stifle our ambitions.

I was the program director back then—a paid staff position that coordinated our schedule and trained our volunteer deejays. We were early adopters of the Internet, posting our schedule online and cultivating a community of listeners before cultivating online communities was cool. We had another staff member that year—the intense and forward-thinking Gavin Shearer (now with Apple), who had the audacious idea that we could expand our coverage area by *broadcasting over the Internet*. Today, that statement probably seems mundane, given contemporary sensibilities and expectations for streaming anything and everything via smart devices. But in 1994 it was absolutely groundbreaking. It was years before streaming audio would become mainstream, and the company that popularized it (RealNetworks) had yet to debut its service.

Our bubble was burst,
our egos deflated. But our
collective sense that we had
done something important and
meaningful never waned. A new
era had begun, and we were
among the earliest pioneers.

Being young, hyper and a little crazy, we decided to go for it. We found a local ISP willing to provide bandwidth, cobbled together a “coalition of the willing” among university administrators to support the project, and then chose the technology that would make it all work: CU-SeeMe, a primitive teleconferencing client available

only on the Mac. We placed a webcam in the studio, pointed it at a fishbowl with our logo superimposed on it, and flipped the switch.

That day we produced our triumphal press release. “KUGS becomes first radio station in the world to broadcast on the Internet.” We sent it out over the AP Wire and became the subject of multiple media inquiries, most of them of the confused variety. “What the hell is Internet broadcasting?” was the most common refrain. We smiled, answered politely, and soaked it in, knowing that we had achieved something momentous and meaningful. Even then, we knew that this would change everything.

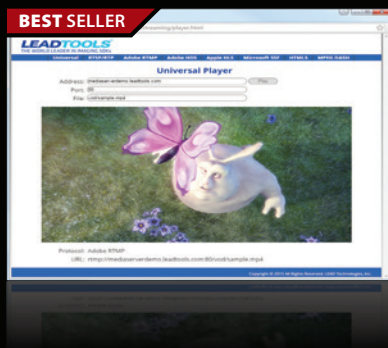
Our moment in the sun was brief. Within 24 hours of our announcement, we found out that WXYC in North Carolina had beaten us to the punch—by a mere two weeks. It turns out we were the second radio station to broadcast on the Internet. And adding insult to injury, WXYC was also 89.3 FM. The jerks stole our frequency! Our bubble was burst, our egos deflated. But our collective sense that we had done something important and meaningful never waned. A new era had begun, and we were among the earliest pioneers. I still get shivers just thinking about it.

KUGS eventually stopped broadcasting via the Internet—the costs were too high, and the benefits eventually determined to be too low. Plus, who needs the Internet when the station transmitter now boasts 950 watts of pure, unadulterated power? Rumor has it that all the notorious dead spots of my earlier days in Bellingham are now covered, no Internet connection required.

I hadn’t thought about all this until recently, when I made a brief stop in Raleigh, N.C., last month. There among the booming tech economy and great BBQ joints were the signs to University of North Carolina, Chapel Hill. For a brief moment I considered an Animal House-style raid where I would tee-pee the rival studio for its unforgivable sin of beating us to the punch by two weeks. Instead, I chose to salute my fellow pioneers from all those years ago.

For now, I simply ask you to remember the intrepid students of Western Washington University and University of North Carolina, Chapel Hill, each time you stream your favorite song on Spotify, or watch “House of Cards” on Netflix. It’s been 20 years since we launched the streaming revolution, and the echoes of those accomplishments still resonate today. ■

KEITH BOYD formerly managed the developer documentation team in the Cloud and Enterprise (C&E) division at Microsoft. Prior to arriving in C&E in 2013, he held the same role in Windows. When he’s not at work, he’s reminiscing about the past while enjoying time with his family, including his wife, Leslie, and three sons, Aiden, Riley and Drew.



LEADTOOLS Multimedia Suite SDK V19

from \$3,495.00 SRP



Create server and workstation applications with Media Streaming, DVR, MPEG-2 TS and more.

- Save one file and stream to any device with the new Media Streaming Server
- Play, convert and capture audio and video streams
- Use MPEG-2 Transport Streams for security monitoring applications and UAV ground stations
- High-performance codecs for H.265, H.264, MPEG-2, MPEG-4, AAC and more
- Includes native DirectShow and Media Foundation libraries for .NET and C/C++



GrapeCity ComponentOne Studio 2015 V2

from \$1,315.60



Empower Your .NET Business Applications with Enterprise-Level Controls.

- Solutions for Data Visualization, Data Management, Reporting, and Business Intelligence
- Hundreds of .NET UI Controls for all Visual Studio Platforms
- New Web API Edition and Visual Studio 2015 Support
- Adaptive, Mobile-Friendly, and Touch-Enabled Controls



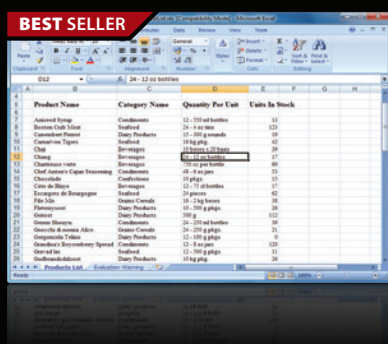
Help & Manual Professional

from \$586.04



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



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



Aspose.Total

Every Aspose component combined in one powerful suite.

Powerful File APIs



Aspose.Cells

XLS, XLSX, XLSM, XLTX, CSV...



Aspose.Email

MSG, EML, PST, EMLX...



Aspose.Words

DOC, DOCX, RTF, HTML, PDF...



Aspose.BarCode

JPG, PNG, BMP, GIF, TIFF, WMF...



Aspose.Pdf

PDF, XML, XLS-FO, HTML, ePUB...



Aspose.Imaging

PDF, BMP, JPG, GIF, TIFF, PNG...



Aspose.Slides

PPT, PPTX, POT, POTX, XPS...



Aspose.Tasks

XML, MPP, SVG, PDF, TIFF, PNG...

... and many more!



Get your FREE evaluation copy at
www.aspose.com



Working with Files?

Try Aspose File APIs

CONVERT
PRINT
CREATE
COMBINE
MODIFY

files from your applications!



Over 15,000 Happy Customers

.NET

Java

Cloud

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



Event Sourcing for the Common Application

It's such a common and natural activity you might not even give it much thought. When you think of data storage, you naturally consider a format that simply preserves the current state of your data. While there are some large-scale systems, such as the insurance or banking industries where any software actions are carefully tracked and recorded, saving the current data state is more than enough for most applications and Web sites.

The current-state approach takes a snapshot of the system state and makes it persistent. The data typically resides in a relational database. That's enough to conduct new transactions and retrieve the results of past transactions. Scenarios where current state storage is insufficient haven't been too common for the past decade.

These days, the business landscape is rapidly changing. Tracking business and domain events is becoming more of a necessity. Event Sourcing (ES) is a pattern that affects storage architecture and the manner in which you retrieve and insert stored data. ES isn't just about auditing and recording business-relevant events in a persisted domain. It's about using a lower abstraction level for saving your data and using ad hoc tools and patterns to create multiple data projections.

ES might look like a smart and cool way to log and audit business functions, but it's really a new storage model theory that's as relevant as the relational model has been since the beginning. It might even have a greater impact on modern software than NoSQL storage. ES isn't an alternative to today's relational and NoSQL products. You implement ES on top of relational databases and NoSQL data stores. ES is about changing your vision of application storage and using events instead of stateful values as the data tokens.

When Does Event Sourcing Help?

A basic but common way to go beyond current state storage is to track update history. Imagine a simple bookstore application. You have a description property on each book and you give your users editing permission. Should you keep track of an old description when a user enters a new one?

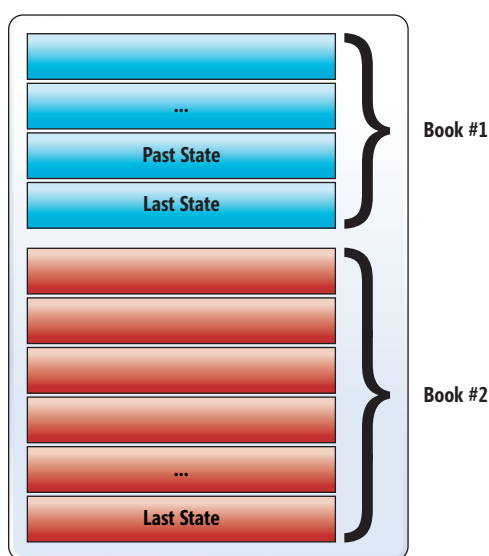


Figure 1 Multiple Records Hold Entity History

Everyone's requirements might be different, but imagine that tracking updates in this example is important. How would you implement that? One option is to store the current book state and log any update details on a separate table. The update record would contain the update delta, such as the old and new value of each modified column.

You could also do this a different way. The Books table could contain multiple records for the same book marked with a given ID. Each record would represent a timestamped state listed in order (see Figure 1).

This scenario requires an ad hoc API to read the current record state. It's not simply a query in the repository that selects the record by ID. You have to pick up the one with the latest timestamp or highest updated

progressive number. Also, the union of all events that relate to a given data entity form a stream. That event stream is a popular concept in ES.

Event Sourcing (ES) is a pattern that affects storage architecture and the manner in which you retrieve and insert stored data.

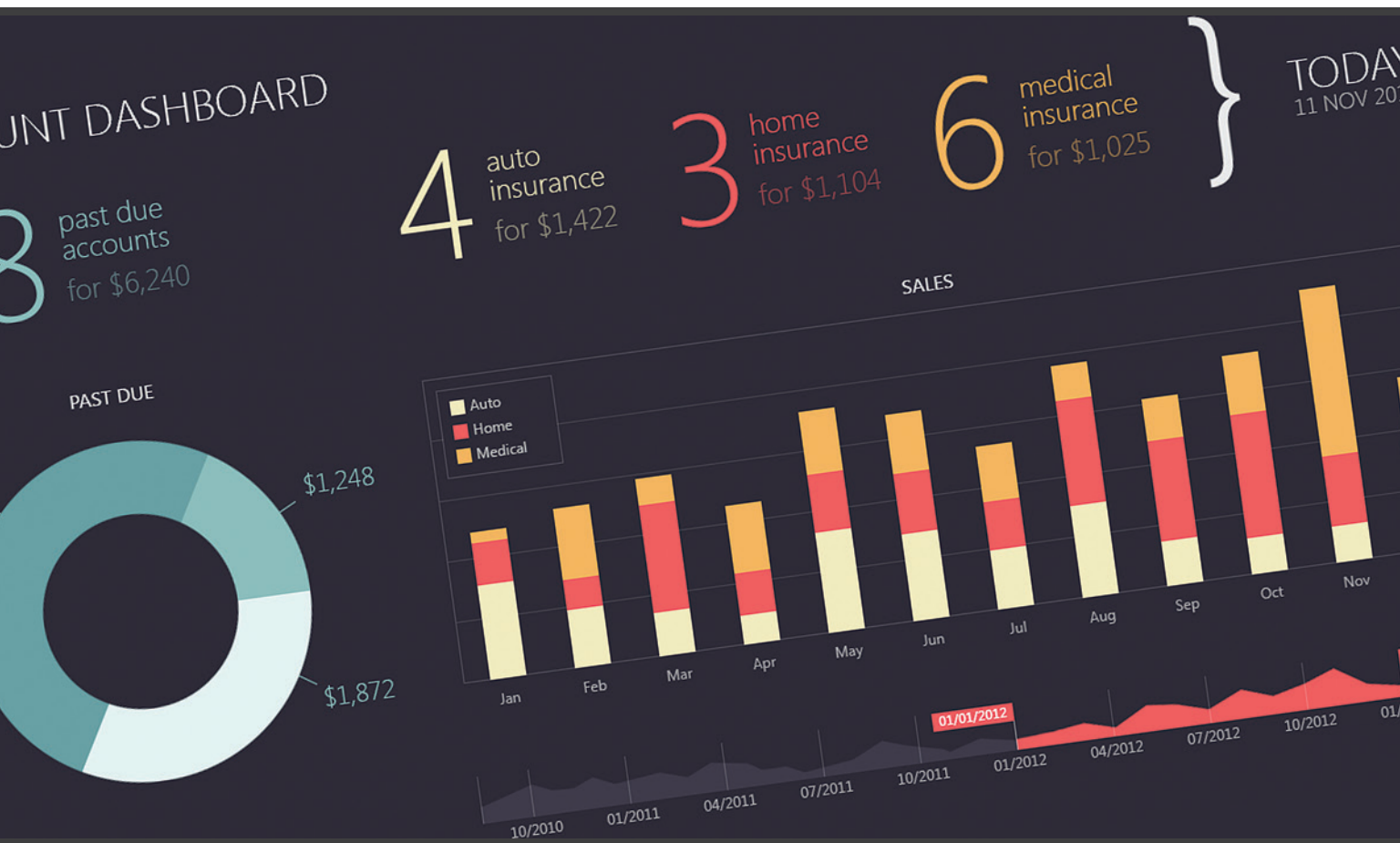
ES helps whenever business demands you track a series of events. While ES might look like cross-cutting concerns such as logging or auditing, it's quite different. It doesn't log events to profile or track exceptions. It just tracks business events. And it isn't a cross-cutting concern, but an architectural decision that applies primarily to storage.

Event Sourcing Defined

In a nutshell, ES is about using events as the primary data source. ES isn't necessarily useful to just any applications, so developers blissfully ignored it for decades. If ES seems useless today, it's mostly because you don't need it yet.

Dashboards & Analytics

DevExpress Universal ships with everything you'll need to create information-rich decision support systems and to distribute your dashboard solutions royalty-free.



Your Next Great Dashboard Starts Here

Learn how you can create fully customizable Dashboards with our flexible data visualization tools.

devexpress.com/dashboard



I like to summarize the need for ES as follows: If a domain expert needs to track the sequence of events the software can produce, then event sourcing is a viable option. Otherwise, it could be events are still useful to express workflows and concatenate pieces of business logic. In this case, though, events aren't first-class citizens in the domain and may not be persisted. This is the mainstream scenario today.

Event Sourcing helps whenever business demands you track a series of events.

Let's see what you do when events are the primary data source of your application. ES affects two aspects of storage: persistence and queries. Persistence is characterized by three core operations—Insert, Update and Delete. In an ES scenario, Insert is nearly the same as in a classic system that persists current entity state. The system receives a request and writes a new event to the store. The event contains a unique identifier (for example, a GUID), type name or code that identifies the type of the event, a timestamp and associated information.

The Update consists of another Insert in the same container of data entities. The new entry simply reflects the data—which properties have changed, the new value and, if relevant in the business domain, why and how it changed. Once an update has been performed, the data store evolves, as shown in **Figure 2**.

The Delete operation works in the same way as an Update, except it has different information so it's clear it was a deletion.

Making updates like this immediately poses a few issues when it comes to queries. How would you know if a given record exists or what its current state might be? That requires an ad hoc layer for queries to conceptually select all records with a matching ID, then analyze the data set event after the event.

For example, it could create a new data entity based on the content of the Created event. Then it would replay all successive steps and return what remains at the end of the stream. This technique is known as event replay. The plain replay of events to rebuild the state might raise some concerns about performance.

Think about a bank account. A customer who opened a bank account years ago would have accumulated hundreds of operations

and events since then. To get the current balance, you have to replay several hundred operations to rebuild the current account state. That might not be always practical.

There are workarounds for this scenario. The most important consists of creating snapshots. A snapshot is a record that saves the known state of the entity at a given time. This way, there's no need to replay events dated before the snapshots.

ES isn't bound to any technology or product, whether a particular relational database or NoSQL data store. ES does raise the need for at least a special software component—the event store. The event store is essentially an event log. You can create one using your own code on top of any data store API of your choice.

An event store has two main characteristics. First and foremost it's an append-only data store. It doesn't support updates and may optionally support for only specific types of deletions. Second, an event store must be able to return the stream of events associated with a given key. You can create this layer of code yourself or use available tools and frameworks.

Event Store Options

You can implement an event store using anything that works. It often uses a relational database or some flavor of a NoSQL data store as the persistence engine. If you do plan to use a relational database, you can have one table per type of entity that produces one row per event.

Events typically have different layouts. For example, each event might have a different number of properties to save, which makes it hard to work out a common schema for all rows. If a common schema resulting from the union of all possible columns is even possible and performs acceptably, this is an easy option to implement.

An event store has two main characteristics.

Otherwise, you can look into the Column Index Store feature of SQL Server 2014, which configures the table to store data in vertical columns instead of horizontal rows. Another option that works with any version of SQL Server is normalizing event properties to a JSON object and storing it as a string in a single column.

In NoSQL jargon, “document” is an object with a variable number of properties. Some NoSQL products specialize in storing documents. From a developer's perspective, it couldn't be easier. Create a class, fill it with values, and store it as is. The type of class is key information that links multiple events. If you go with NoSQL, then you just have an event object and save it.

Ongoing Projects

ES is a relatively young architectural approach. The standard tools that help write code on top of event-based data stores are still emerging. You can definitely arrange an ES solution on your own, but some ad hoc tools can help you deal with storage of events in a more structured way.

The primary benefit of using an event-aware data store is the tool, like a database, guarantees you only perform actions that read and

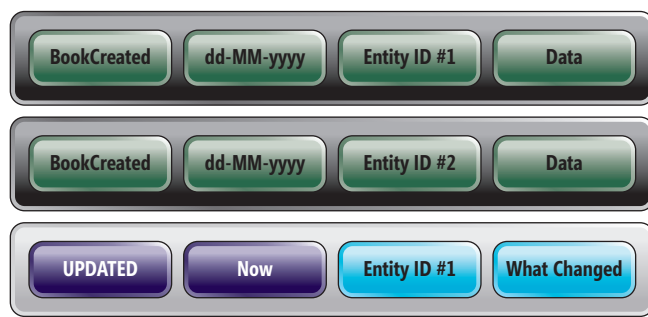


Figure 2 A New Record Indicates Update to Entity with ID #1

append events in a way that guarantees the business consistency of the event-sourcing approach. One framework specifically

Event Sourcing uses events as the application data source.

designed for storing events is the NEventStore project (neventstore.org). This lets you write and read back events, and operates independent of persistence. Here's how you save an event:

```
var store = Wireup.Init()
    .UsingSqlPersistence("connection")
    .InitializeStorageEngine()
    .UsingJsonSerialization()
    .Build();
var stream = store.CreateStream(aggregateId);
stream.Add(new EventMessage { Body = eventToSave });
stream.CommitChanges(aggregateId);
```

To read events back, open the stream and loop through the collection of committed events.

Event Store (geteventstore.com) is another that works by offering an API for plain HTTP and .NET for event streams. In ES jargon, an aggregate equates a stream in the store. You can perform three basic operations on an event stream: write events; read the last event, a specific event and even a slice of events; and subscribe to get updates.

There are three types of subscriptions. One is volatile, which means writing an event to a given stream invokes a callback function is invoked every time. Another is catch-up, which means you'll get notifications for each event in the store starting from a given one and after that for any newly added event. Finally, the persistent subscription addresses the scenario when multiple consumers are waiting for events to process. The subscription guarantees events are delivered to consumers at least once, but possibly multiple times and in unpredictable order.

Wrapping Up

Event Sourcing uses events as the application data source. You don't architect the application to save the last-known state of entities, but the list of relevant business events. The event data source stores data at a low level of abstraction. You need to apply projections to get from there to the actual entity state required for transactions and queries. The projection is the process of replaying events and performing some tasks. The most obvious projection is building the current state; but you can have any number or type of projections out of events. ■

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

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)



Classy Types in the Windows Runtime

The Windows Runtime (WinRT) allows component developers to present a classy type system to app developers. Given that the Windows Runtime is implemented entirely with COM interfaces, this might seem quite absurd to a developer familiar with C++ and classic COM. COM is, after all, a programming model centered on interfaces rather than classes. The COM activation model allows classes to be constructed with `CoCreateInstance` and friends, but this can plausibly support only something akin to a default constructor. The WinRT `RoActivateInstance` does nothing to change this perception. So how does it work?

It isn't actually as incongruous as it might at first appear. The COM activation model already provides the fundamental abstractions necessary for supporting a classy type system. It merely lacks the metadata necessary to describe it to a consumer. The COM activation model defines class objects and instances. An instance of some class is never created directly by a consumer. Even if an app developer calls `CoCreateInstance` to create some instance of a class, the OS still has to get the class object in order to retrieve the desired instance.

The Windows Runtime calls class objects by a new name but the mechanics are the same. An activation factory is a consumer's first port of call into a component. Rather than implementing `DllGetClassObject`, a component exports a function called `DllGetActivationFactory`. The way that classes are identified has changed, but the result is just an object that might be queried further in connection with the given class. A traditional class object, henceforth called an activation factory, might have implemented the `IClassFactory` interface that allows the caller to create instances of said class. In the new world, activation factories in the Windows Runtime must implement the new `IActivationFactory` interface—effectively providing the same service.

As I explored in the last two installments of my column, a runtime class in the Windows Runtime that's decorated with the `activatable` attribute produces metadata that instructs a language projection to permit default construction. The assumption is that the activation factory for the attributed class will provide such a default constructed object via `IActivationFactory`'s `ActivateInstance` method. Here's a simple runtimeclass in IDL that represents a class with a default constructor:

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

Additional constructors may also be described in IDL with, you guessed it, interfaces housing collections of methods that represent additional constructors with different sets of parameters. Just

as the `IActivationFactory` interface defines default construction, component- and class-specific interfaces might describe parameterized construction. Here's a simple factory interface for creating Hen objects given a specific number of clucks:

```
runtimeclass Hen;

[version(1)]
[uuid(4fa3a693-6284-4359-802c-5c05afa6e65d)]
interface IHenFactory : IInspectable
{
    HRESULT CreateHenWithClucks([in] int clucks,
                                [out,retval] Hen ** hen);
}
```

Like `IActivationFactory`, the `IHenFactory` interface must inherit directly from `IInspectable` for no apparent reason. Each of this factory interface's methods is then projected as an additional constructor with the given parameters, and each must conclude with a logical return value whose type is that of the class. This factory interface is then explicitly associated with the runtime class via another activatable attribute indicating the interface name:

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

If a default constructor makes no sense for your class then simply omit the original `activatable` attribute, and language projections of your class will likewise lack a default constructor. What if I ship a version of my poultry component and then realize it lacks the ability to create hens with large combs? I certainly can't change the `IHenFactory` interface now that I've shipped it to customers because COM interfaces are necessarily immutable binary and semantic contracts. No problem, I can simply define another interface housing any additional constructors:

```
[version(2)]
[uuid(9fc40b45-784b-4961-bc6b-0f5802a4a86d)]
interface IHenFactory2 : IInspectable
{
    HRESULT CreateHenWithLargeComb([in] float width,
                                    [in] float height,
                                    [out, retval] Hen ** hen);
}
```

I can then associate this second factory interface with the Hen class as before:

```
[version(1)]
[activatable(1)]
[activatable(IHenFactory, 1)]
[activatable(IHenFactory2, 2)]
runtimeclass Hen
{
    [default] interface IHen;
```


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

Language projections take care of the amalgamation and the app developer simply sees a number of constructor overloads, as depicted in **Figure 1**.

The component developer is careful to implement these factory interfaces, along with the prerequisite `IActivationFactory` interface:

```
struct HenFactory : Implements<IActivationFactory,
                           ABI::Sample::IHenFactory,
                           ABI::Sample::IHenFactory2>
{
};
```

Here again I'm relying on the `Implements` class template I described in my December 2014 column (msdn.microsoft.com/magazine/dn879357). Even if I chose to prohibit default construction, the hen's activation factory would still need to implement the `IActivationFactory` interface because the `DllGetActivationFactory` function that components must implement is hardcoded to return an `IActivationFactory` interface. This means that a language projection must first call `QueryInterface` on the resulting pointer if the app needs anything other than default construction. Still, an implementation of `IActivationFactory`'s `ActivateInstance` virtual function is required, but a no-op such as this will suffice:

```
virtual HRESULT __stdcall ActivateInstance(IInspectable ** instance)
noexcept override
{
    *instance = nullptr;
    return E_NOTIMPL;
}
```

The additional construction methods can be implemented in whatever way makes sense for the given implementation, but a simple solution would be to simply forward the arguments to the internal class itself. Something like this:

```
virtual HRESULT __stdcall CreateHenWithClucks(int clucks,
                                             ABI::Sample::IHen ** hen)
noexcept override
{
    *hen = new (std::nothrow) Hen(clucks);
    return *hen ? S_OK : E_OUTOFMEMORY;
}
```

This assumes the C++ `Hen` class doesn't have a throwing constructor. I might need to allocate a C++ vector of clucks in that constructor. In that case, a simple exception handler will suffice:

```
try
{
    *hen = new Hen(clucks);
    return S_OK;
}
catch (std::bad_alloc const &)
{
    *hen = nullptr;
    return E_OUTOFMEMORY;
}
```

What about static class members? You shouldn't be surprised to learn that this, too, is implemented with COM interfaces on the activation factory. Back in IDL, I can define an interface to house any and all static members. How about something to report on the number of layer hens in the backyard:

```
[version(1)]
[uuid(60086441-fcbb-4c42-b775-88832cb19954)]
interface IHenStatics : IInspectable
{
    [propget] HRESULT Layers([out, retval] int
    * count);
}
```

I then need to associate this interface with my same runtime class with the static attribute:

```
[version(1)]
[activatable(1)]
[activatable(IHenFactory, 1)]
[activatable(IHenFactory2, 2)]
[static(IHenStatics, 1)]
runtimeclass Hen
{
    [default] interface IHen;
}
```

The implementation in C++ is equally straightforward. I'll update the `Implements` variadic template as follows:

```
struct HenFactory : Implements<IActivationFactory,
                           ABI::Sample::IHenFactory,
                           ABI::Sample::IHenFactory2,
                           ABI::Sample::IHenStatics>
{
};
```

And provide a suitable implementation of the `get_Layers` virtual function:

```
virtual HRESULT __stdcall get_Layers(int * count) noexcept override
{
    *count = 123;
    return S_OK;
}
```

I'll leave it to you to provide a more accurate head count ... I mean hen count.

On the consumer side, inside the app, it all appears very simple and concise. As illustrated in **Figure 1**, the IntelliSense experience is quite helpful. I can use the `CreateHenWithClucks` constructor as if it were just another constructor on a C# class:

```
Sample.Hen hen = new Sample.Hen(123); // Clucks
```

Of course, the C# compiler and Common Language Runtime (CLR) have a fair amount of work to do in order for this to run. At run time, the CLR calls `RoGetActivationFactory`, which calls `LoadLibrary` followed by `DllGetActivationFactory` to retrieve the `Sample.Hen` activation factory. It then calls `QueryInterface` to retrieve the factory's `IHenFactory` interface implementation, and only then can it call the `CreateHenWithClucks` virtual function to create the `Hen` object. This is not to mention the many, many additional calls to `QueryInterface` the CLR insists on as it prods each and every object that enters the CLR to discover various attributes and semantics of said objects.

And calling a static member appears equally simple:

```
int layers = Sample.Hen.Layers;
```

But here again the CLR must call `RoGetActivationFactory` to get the activation factory, followed by `QueryInterface` to retrieve the `IHenStatics` interface pointer before it can call the `get_Layers` virtual function to retrieve this static property's value. The CLR does, however, cache activation factories so the cost is somewhat amortized across many such calls. On the other hand, it also makes various attempts at factory caching within a component rather pointless and unnecessarily complex. But that's a topic for another day. Join

me next month as we continue to explore the Windows Runtime from C++.

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**

```
public void SetWindow(CoreWindow window)
{
    int layers = Sample.Hen.Layers;

    var hen = new Sample.Hen();
    hen.Cluck(window);
}
```

Figure 1 IDL-Defined Factory Methods in C#

NEXT GENERATION 1&1 CLOUD SERVER

TRY IT FOR 1 MONTH - FREE!

Then starting at \$9.99 per month*

Powered by  Cloud
Technology

Top performer


CLOUD
SPECTATOR

EASY TO USE – READY TO GO

The new 1&1 Cloud Server offers all the advantages of dedicated hardware performance combined with the flexibility of the cloud!

FLEXIBLE & AFFORDABLE

Customized configuration

- SSD, RAM and CPU can be adjusted independently, flexibly and precisely



- **NEW:** Pre-configured packages available



Transparent costs

- Billing by the minute
- Clearly structured cost overview enables efficient planning and management
- No minimum contract term

EASY & SECURE

1&1 Cloud Panel

- Innovative, user-friendly interface with smart administration

Security

- Built-in firewall to protect your server from online threats
- Backups and snapshots to prevent accidental data loss
- High-performance 1&1 Data Centers are among the safest in the US

ALL-INCLUSIVE

Best performance

- Unlimited traffic
- Premium SSD with the highest performance
- Private networks, professional API, load balancers, firewalls and more – all easy to configure
- Ready-to-use applications including WordPress, Drupal™ and Magento®
- Powered by Intel® Xeon® Processor E5-2683 V3 (35M Cache, 2.00 Ghz)



☎ 1 (877) 461-2631



1and1.com

*1&1 Cloud Server is available free for one month, after which regular price of \$9.99/month applies. No setup fee is required. Visit 1and1.com for full offer details, terms and conditions. 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.



Revisiting JavaScript Data Binding— Now with Aurelia

I've never been much of a front-end developer, but every once in a while I have a reason to play with a UI. In my June 2012 column, after seeing a user group presentation on Knockout.js, I dug in and wrote an article about data binding OData in a Web site using Knockout (msdn.microsoft.com/magazine/jj133816). A few months later, I wrote about adding Breeze.js into the mix to make the data binding with Knockout.js even easier (msdn.microsoft.com/magazine/jj863129). When I wrote about modernizing an old ASP.NET 2.0 Web Forms app and using Knockout again in 2014, I got teased by some friends who said that Knockout was “so 2012.” Newer frameworks such as Angular also did data binding and so much more. But I wasn't really interested in the “so much more,” so Knockout was fine for me.

Well, now it's 2015 and while Knockout is still valid and relevant and awesome for JavaScript data binding, I wanted to spend some time with one of the new frameworks and chose Aurelia (Aurelia.io) because I know so many Web developers who are excited about it. Aurelia was started by Rob Eisenberg, who was behind Durandal, another JavaScript client framework, though he stopped production on that to go work on the Angular team at Google. Eventually he left Angular and, rather than reviving Durandal, created Aurelia from the ground up. There are a lot of interesting things about Aurelia. I have a lot more to learn for sure, but want to share with you some of the data-binding tricks I learned, as well as a bit of trickery with EcmaScript 6 (ES6), the latest version of JavaScript, which became a standard in June 2015.

An ASP.NET Web API to Serve Data to My Web Site

I'm using an ASP.NET Web API I built to expose data I'm persisting with Entity Framework 6. The Web API has a handful of simple methods to be called via HTTP.

Figure 1 The Get Method from the Web API

```
public IEnumerable<ViewListNinja> Get(string query = "",
    int page = 0, int pageSize = 20)
{
    var ninjas = _repo.GetQueryableNinjasWithClan(query, page, pageSize);
    return ninjas.Select(n => new ViewListNinja
    {
        ClanName = n.Clan.ClanName,
        DateOfBirth = n.DateOfBirth,
        Id = n.Id,
        Name = n.Name,
        ServedInOniwaban = n.ServedInOniwaban
    });
}
```

Code download available at msdn.microsoft.com/magazine/msdnmag0915.

The Get method, shown in **Figure 1**, takes in some query and paging parameters and passes them on to a repository method that retrieves a list of Ninja objects and their related Clan objects using an Entity Framework DbContext. Once the Get method has the results in hand, it transforms those results into a set of ViewListNinja Data Transfer Objects (DTOs), defined elsewhere. This is an important step because of the way JSON is serialized, which is a bit overzealous with circular references from the Clan back to other Ninjas. With the DTOs, I avoid a wasteful amount of data going over the wire and I get to shape the results to more closely match what's on the client.

The Aurelia paradigm pairs one view model (a JavaScript class) with one view (an HTML file) and performs data binding between the two.

Figure 2 shows a view of the JSON results from that method based on a query that retrieved two Ninja objects.

Querying the Web API Using the Aurelia Framework

The Aurelia paradigm pairs one view model (a JavaScript class) with one view (an HTML file) and performs data binding between the two. Therefore, I have a ninjas.js file and a ninjas.html file. The Ninjas view model is defined as having an array of Ninjas, as well as a Ninja object:

```
export class Ninja {
    searchEntry = '';
    ninjas = [];
    ninjaId = '';
    ninja = '';
    currentPage = 1;
    textShowAll = 'Show All';

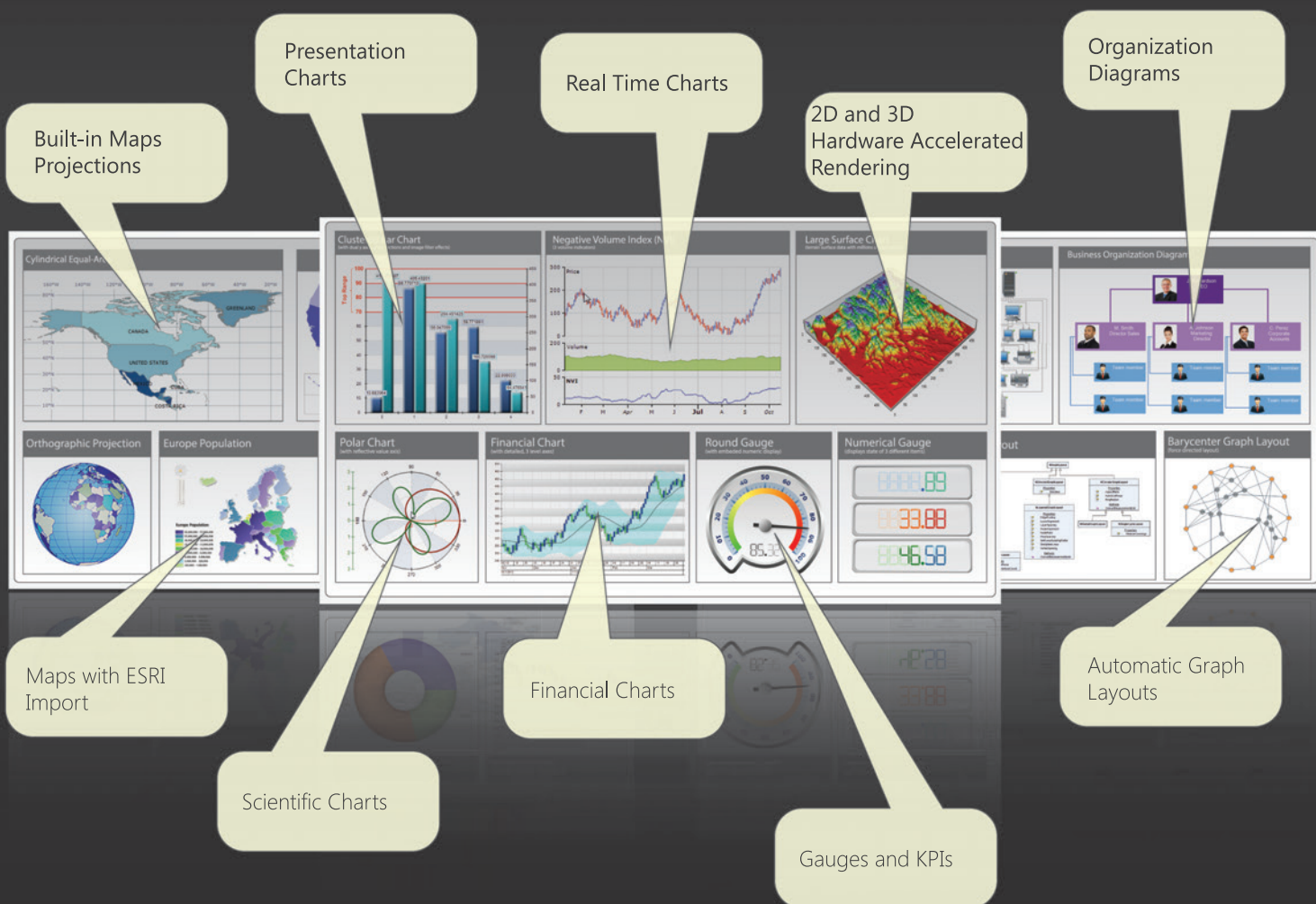
    constructor(http) {
        this.http = http;
    }
}
```

The most critical method in ninjas.js is retrieveNinjas, which calls the Web API:

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 an e-mail to support@nevron.com.


```

retrieveNinjas() {
  return this.http.createRequest(
    "/ninjas/?page=" + this.currentPage +
    "&pageSize=100&query=" + this.searchEntry)
    .asGet().send().then(response => {
      this.ninjas = response.content;
    });
}

```

Elsewhere in the Web app I've set up the base URL that Aurelia will find and incorporate it into the request's URL:

```

x.withBaseUrl('http://localhost:46534/api');

```

It's worth noting that the ninjas.js file is just JavaScript. If you've used Knockout, you might recall that you have to set up the view model using Knockout notation so that when the object is bound to the markup, Knockout will know what to do with it. That's not the case for Aurelia.

The response now includes the list of Ninjas and I set that to the ninjas array of my view model, which gets returned to the ninjas.html page that triggered the request. There's nothing in the markup that identifies the model—that's taken care of because the model is paired with the HTML. In fact, most of the page is standard HTML and some JavaScript, with just a few special commands that Aurelia will find and handle.

Data Binding, String Interpolation and Formatting

The most interesting part of ninjas.html is the div, which is used to display the list of ninjas:

```

<div class="row">
  <div repeat.for="ninja of ninjas">
    <a href="#/ninjas/${ninja.Id}" class="btn btn-default btn-sm">
      <span class="glyphicon glyphicon-pencil" /> </a>
    <a click.delegate="${parent.deleteView(ninja)}" class="btn btn-default btn-sm">
      <span class="glyphicon glyphicon-trash" /> </a>
    ${ninja.Name} ${ninja.ServedInOniwaban ? '[Oniwaban]': ''}
    Birthdate:${ninja.DateOfBirth | dateFormat}
  </div>
</div>

```

The first Aurelia-specific markup in that code is *repeat.for* “*ninja of ninjas*,” which follows an ES6 paradigm for looping. Because Aurelia comprehends the view model, it understands that “*ninjas*” is the property defined as an array. The variable “*ninja*” can be any name, for example “*foo*.” It just represents each item in the ninjas array. Now it's just a matter of iterating through the items in the ninjas array.

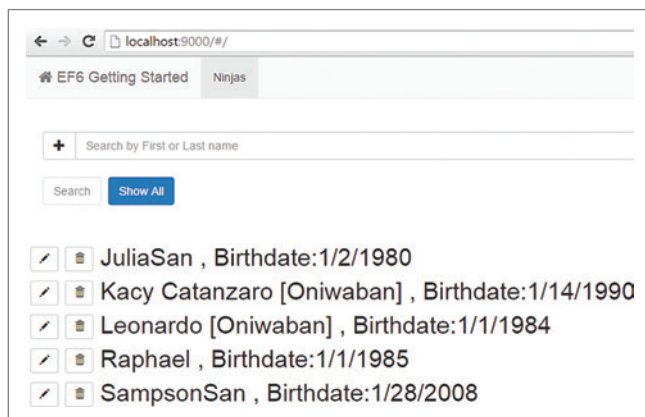


Figure 3 Displaying All Ninjas with Properties Bound One Way by Aurelia via String Interpolation

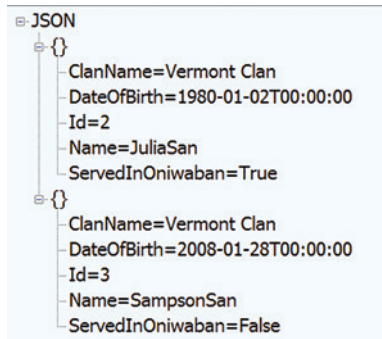


Figure 2 JSON Results from Web API Request for List of Ninjas

Skip down to the markup where the properties are displayed, for example “*\${ninja.Name}*.” This is an ES6 feature that Aurelia takes advantage of, which is called string interpolation. String interpolation makes it easier to compose strings with variables embedded in them rather than, for example, by concatenating. So with a variable name=“Julie,” I can write in JavaScript:

```

`Hi, ${name}!`

```

and it will be processed as “Hi, Julie!” Aurelia takes advantage of ES6 string interpolation and infers one-way data binding when it encounters that syntax. So that last line of code, beginning with *\${ninja.Name}* will output the

ninja properties along with the rest of the HTML text. If you code in C# or Visual Basic, it's worth noting that string interpolation is a new feature of both C# 6.0 and Visual Basic 14.

I did have to learn a bit more JavaScript syntax along the way, for example the conditional evaluation of the ServedInOniwaban Boolean, which turns out to have the same syntax as C#—*condition ? true : false*.

Aurelia takes advantage of ES6 string interpolation and infers one-way data binding when it encounters that syntax.

The date formatting I've applied to the DateOfBirth property is another Aurelia feature and may be familiar to you if you've used XML. Aurelia uses value converters. I like to use the moment JavaScript library to help with date and time formatting and am taking advantage of that in the date-format.js class:

```

import moment from 'moment';

export class dateFormatValueConverter {
  toView(value) {
    return moment(value).format('M/D/YYYY');
  }
}

```

Keep in mind that you do need “ValueConverter” in the class name.

At the top of the HTML page, just under the initial *<template>* element, I have a reference to that file:

```

<template>
  <require from="./date-format"></require>

```

Now the string interpolation is able to find the dateFormat [ValueConverter] in my markup and apply it to the output, as shown in Figure 3.

I want to point out another instance of binding in the div, but it's event binding, not data binding. Notice that in the first hyperlink tag I use a common syntax, embedding a URL in the href attribute. In the second, however, I don't use href. Instead, I use click.delegate. Delegate is not a new command, but Aurelia does handle it in a special way that's much more powerful than a standard onclick event handler. Read more at bit.ly/1Jv38Z. I'll continue to focus on the data-related binding here.

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](http://www.DynamicPDF.com)



TRY OUR PDF SOLUTIONS FREE TODAY!

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

ceTe software

The edit icons lead to a URL that includes the ninja's ID. I've instructed the Aurelia routing mechanism to route to a page called Edit.html. This is tied to a view model that's in a class named Edit.js.

The most critical methods of Edit.js are for retrieving and saving the selected ninja. Let's start with retrieveNinja:

```
retrieveNinja(id) {
    return this.http.createRequest("/ninjas/" + id)
        .asGet().send().then(response => {
            this.ninja = response.content;
        });
}
```

This builds a similar request to my Web API as before, but this time appending the id to the request.

In the ninjas.js class I bound the results to a ninjas array property of my view model. Here I'm setting the results, a single object, to the ninja property of the current view model.

Here is the Web API method that will get called because of the id that's appended to the URI:

```
public Ninja Get(int id)
{
    return _repo.GetNinjaWithEquipmentAndClan(id);
}
```

The results from this method are much richer than those returned for the ninja list. **Figure 4** shows the JSON returned from one of the requests.

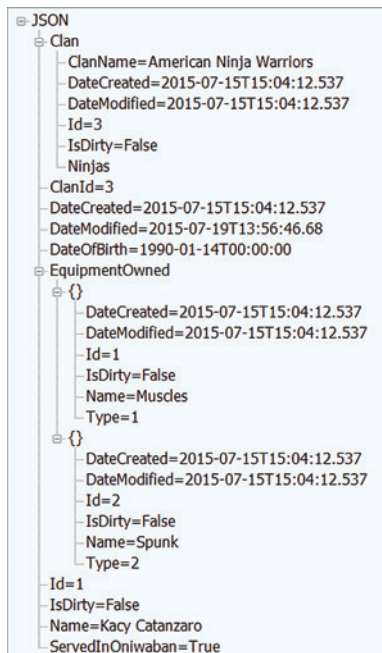


Figure 4 JSON Results of a WebAPI Request for a Single Ninja

alongside browsers and libraries as they evolve over time. You can read more about the adaptive binding at bit.ly/1GhDCDB.

Currently, you can only edit the ninja name, birth date and Oniwaban indicator. When a user unchecks Served in Oniwaban and hits the Save button, this action calls into my view model save method, which does something interesting before pushing the data back to my Web API. Currently, as you saw in **Figure 4**, the ninja object is a deep graph. I don't need to send all of that back for saving, just the relevant properties. Because I'm using EF on the other end, I want to be sure the properties I didn't edit also go back so they don't get replaced with null values in the database. So I'm creating an on-the-fly DTO called ninjaRoot. I've already declared ninjaRoot as a property of my view model. But the definition of ninjaRoot will be indicated

by how I build it in my Save method (see **Figure 6**). I've been careful to use the same property names and casing that my WebAPI expects so it can deserialize this into the known Ninja type in the API.

Notice the "asPost" method in this call. This ensures the request goes to the Post method in my Web API:

```
public void Post([FromBody] object ninja)
{
    var asNinja =
        JsonConvert.DeserializeObject<Ninja>
            (ninja.ToString());
    _repo.SaveUpdatedNinja(asNinja);
}
```

The JSON object is deserialized into a local Ninja object and then passed on to my repository method, which knows to update this object in the database.

When I return to the Ninjas list on my Web site, the change is reflected in the output.

The most critical methods of Edit.js are for retrieving and saving the selected ninja.

Once I've pushed the results into my view model, I can bind the properties of the ninja to elements in the HTML page. This time I use the .bind command. Aurelia infers if something should be bound one-way or two-way or some other way. In fact, as you saw in ninjas.html, it used its underlying binding workflow when presented with the string interpolation. There it used one-time, one-way binding. Here, because I'm using the .bind command and am binding to input elements, Aurelia infers that I want two-way binding. That's its default choice, which I could override by using .one-way or another command in place of .bind.

For brevity, I'll extract just the relevant markup rather than the surrounding elements.

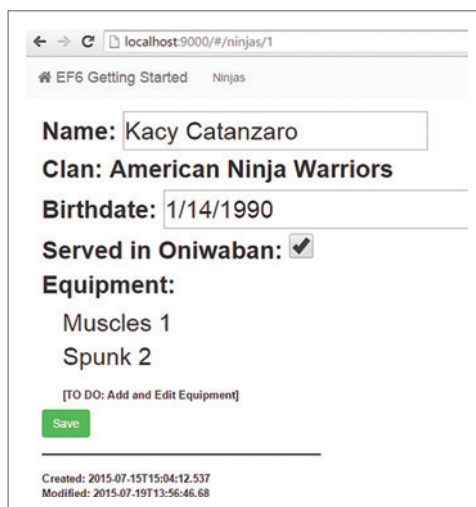


Figure 5 The Edit Page Displaying the Equipment List

Figure 6 The Save Method in the Edit Model-View

```
save() {  
    this.ninjaRoot = {  
        Id: this.ninja.Id,  
        ServedInOniwaban: this.ninja.ServedInOniwaban,  
        ClanId: this.ninja.ClanId,  
        Name: this.ninja.Name,  
        DateOfBirth: this.ninja.DateOfBirth,  
        DateCreated: this.ninja.DateCreated,  
        DateModified: this.ninja.DateModified  
    };  
    this.http.createRequest("/ninjas/")  
        .asPost()  
        .withHeader('Content-Type', 'application/json; charset=utf-8')  
        .withContent(this.ninjaRoot).send()  
        .then(response => {  
            this.myRouter.navigate('ninjas');  
        }).catch(err => {  
            console.log(err);  
        });  
}
```

More Than Just Data Binding

Keep in mind that Aurelia is a much richer framework than for just data binding, but true to my nature, I focused on data in my first steps to explore this tool. You can learn so much more at the Aurelia Web site, and there's a thriving community at gitter.im/Aurelia/Discuss.

I want to proffer a nod of thanks to the tutaurelia.net Web site, a blog series about combining ASP.NET Web API and Aurelia. I leaned on author Bart Van Hoey's Part 6 for my first pass at displaying the list of ninjas. Perhaps by the time this article is published, there will be more posts added to the series.

Once I've pushed the results into my view model, I can bind the properties of the ninja to elements in the HTML page.

The download for this article will include both the Web API solution and the Aurelia Web site. You can use the Web API solution in Visual Studio. I've built it in Visual Studio 2015 using Entity Framework 6 and the Microsoft .NET Framework 4.6. If you want to run the Web site, you'll need to visit the Aurelia.io site to learn how to install Aurelia and run the Web site. You can also see me demonstrate this application in my Pluralsight course, "Getting Started with Entity Framework 6," at bit.ly/PS-EF6Start. ■

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 julieme.me/PS-Videos.

THANKS to the following technical expert for reviewing this article:
Rob Eisenberg (Durandal Inc.)

msdnmagazine.com

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



Visual Studio
EXPERT SOLUTIONS FOR .NET DEVELOPERS



ORLANDO
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

NOV
16-20



Code In The Sun

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



Magenic



PLATINUM SPONSORS



PLURALSIGHT

GOLD SPONSOR

GitHub

SUPPORTED BY



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 SEPTEMBER
16 AND SAVE \$400!**



Use promo code
L360SEP2 by September 16

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

Take the Tour



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 
TRAINING FOR COLLABORATION

SQL Server 
TRAINING FOR DBAs AND IT PROS

ModernApps 
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!

TURN THE PAGE FOR
MORE EVENT DETAILS.



Check Out the Additional Sessions for Devs, IT Pros, & DBAs at Live! 360



SharePoint **LIVE!** TRAINING FOR COLLABORATION

SharePoint Live! features 12+ developer sessions, including:

- Workshop: Developer OnRamp to the Office 365 & Azure AD Highway! - Andrew Connell
- An Introduction to SharePoint 2013 Add-ins for Developers - Rob Windsor
- SharePoint Developers, Come to the Client Side (We Have Cookies) - Bill Ayers
- SharePoint Development Lifecycle for Solutions and Add-ins - Robert Bogue
- Building Office Add-Ins with Visual Studio - Bill Ayers
- Utilizing jQuery in SharePoint - Get More Done Faster - Mark Rackley
- Using Azure to Replace Server-Side Code in Office 365 - Paul Schaefflein



SQL Server **LIVE!** TRAINING FOR DBAs AND IT PROS

SQL Server Live! features 15+ developer sessions, including:

- Encrypting Data in SQL Server - David Dye
- Kick Start! SQL Server 2014 Performance Tips and Tricks - Pinal Dave
- Database Development with SQL Server Data Tools - Leonard Lobel
- Exploring T-SQL Enhancements: Windowing and More - Leonard Lobel
- Python and R for SQL and Business Intelligence Professionals - Jen Stirrup



TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

TechMentor features IT Pro and DBA sessions, including:

- Workshop: Windows PowerShell Scripting and Toolmaking (BYOL-RA) - Jeffery Hicks
- Your Approach to BYOD Sucks! - Simon May
- Windows Server vNext: Here's What We Know - Don Jones
- Ethical Hacking: Methodologies and Fundamentals - Mike Danseglio & Avril Salter
- DISCUSSION: Career Strategies for the IT Pro - Greg Shields
- PowerShell Unplugged: Stuff I Stole from Snover - Don Jones
- Workshop: SQL Server 2014 for Developers - Leonard Lobel

LOOK FOR THE FULL LIVE! 360 PULL-OUT AGENDA ON PAGE 33

Featured Live! 360 Speakers



ANDREW BRUST



MIGUEL CASTRO



ANDREW
CONNELL



DON JONES



DEBORAH
KURATA



ROCKFORD
LHOTKA



MATTHEW
MCDERMOTT



TED NEWARD



JOHN PAPA



BRIAN RANDELL



RACHEL REESE



GREG SHIELDS



Look for the **FULL Live! 360 Pull-Out Agenda on Page 33**


AGENDAS AT-A-GLANCE VISUAL STUDIO LIVE! & MODERN APPS LIVE!

Cloud Computing	Database and Analytics	Lessons Learned and Advanced Practices	Mobile Client	Visual Studio / .NET Framework	Web Development	Windows Client	Modern Apps Live! Sponsored by: Magenic
-----------------	------------------------	--	---------------	--------------------------------	-----------------	----------------	---

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Pre-Conference: Sunday, November 15, 2015					
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:00pm - Meet at Conference Registration Desk to walk over with the group					

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Pre-Conference Workshops: Monday, November 16, 2015			
8:00 AM	5:00 PM	VSM01 Workshop: Service Oriented Technologies: Designing, Developing, & Implementing WCF and the Web API - Miguel Castro	VSM02 Workshop: Triple D: Design, Development, and DevOps - Billy Hollis and Brian Randell	VSM03 Workshop: Busy Developer's Guide to MEANJS - Ted Neward	MAM01 Workshop: Modern App Technology Overview - Android, iOS, Cloud, and Mobile Web - Nick Landry, Kevin Ford, & Steve Hughes
5:00 PM	6:00 PM	EXPO Preview			
6:00 PM	7:00 PM	Live! 360 Keynote: Microsoft 3.0: New Strategy, New Relevance - Pacifica 6 Mary Jo Foley, Journalist and Author; with Andrew Brust, Senior Director, Datameer			

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 1: Tuesday, November 17, 2015					
8:00 AM	9:00 AM	Visual Studio Live! & Modern Apps Live! Keynote: The Future of Application Development - Visual Studio 2015 and .NET 2015 <i>Jay Schmelzer, Director of Program Management, Visual Studio Team, Microsoft</i>					
9:00 AM	9:30 AM	Networking Break • Visit the EXPO - Pacifica 7					
9:30 AM	10:45 AM	VST01 AngularJS 101 - Deborah Kurata	VST02 A Tour of Azure for Developers - Adam Tuliper	VST03 Busy Developer's Guide to NoSQL - Ted Neward	VST04 Visual Studio, TFS, and VSO in 2015 - What's New? - Brian Randell	MAT01 Defining Modern App Development - Rockford Lhotka	
11:00 AM	12:15 PM	VST05 From ASP.NET Site to Mobile App in About an Hour - Ryan J. Salva	VST06 Introduction to Next Generation of Azure PaaS - Service Fabric and Containers - Vishwas Lele	VST07 Real World SQL Server Data Tools (SSDT) - Benjamin Day	VST08 Automate Your Builds with Visual Studio Online or Team Foundation Server - Tiago Pascoal	MAT02 Modern App Architecture - Brent Edwards	
12:15 PM	2:00 PM	Lunch • Visit the EXPO - Oceana Ballroom / Pacifica 7					
2:00 PM	3:15 PM	VST09 I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(Me), Maybe? - Rachel Appel	VST10 Cloud or Not, 10 Reasons Why You Must Know "Websites" - Vishwas Lele	VST11 Windows 10 for Developers: What's New in Universal Apps - Nick Landry	VST12 Defensive Coding Techniques in C# - Deborah Kurata	MAT03 ALM with Visual Studio Online (TFS) and Git - Brian Randell	
3:15 PM	4:15 PM	Networking Break • Visit the EXPO - Pacifica 7					
4:15 PM	5:30 PM	VST13 Better Unit Tests through Design Patterns for ASP MVC, WebAPI, and AngularJS - Benjamin Day	VST14 Running ASP.NET Cross Platform with Docker - Adam Tuliper	VST15 Build Your First Mobile App in 1 Hour with Microsoft App Studio - Nick Landry	VST16 Putting CodedUI Tests on Steroids - Donovan Brown	MAT04 Reusing Logic Across Platforms - Kevin Ford	
5:30 PM	7:30 PM	Exhibitor Reception					

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 2: Wednesday, November 18, 2015					
8:00 AM	9:00 AM	Live! 360 Keynote: DevOps: What it Means to You - Pacifica 6 - Sponsored By PLURALSIGHT Don Jones, Curriculum Director for IT Pro Content, Pluralsight & Brian Randell, Partner, MCW Technologies					
9:15 AM	10:30 AM	VSW01 Mobile App Development with Xamarin and F# - Rachel Reese	VSW02 Notify Your Millions of Users with Notification Hubs - Matt Milner	VSW03 Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - Billy Hollis	VSW04 To Git or Not to Git for Enterprise Development - Benjamin Day	MAW01 Coding for Quality and Maintainability - Jason Bock	
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - Pacifica 7					
11:00 AM	12:15 PM	VSW05 Automated UI Testing for Android and iOS Mobile Apps - James Montemagno	VSW06 Busy Developer's Guide to the Clouds - Ted Neward	VSW07 Designing and Building UX for Finding and Visualizing Data in XAML Applications - Billy Hollis	VSW08 Anything C# Can Do, F# Can Do Better - Rachel Appel & Rachel Reese	MAW02 Start Thinking Like a Designer - Anthony Handley	
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO - Oceana Ballroom / Pacifica 7					
1:45 PM	3:00 PM	VSW09 Stop Creating Forms In Triplicate - Use Xamarin Forms - Matt Milner	VSW10 To Be Announced	VSW11 Developing Awesome 3D Games with Unity and C# - Adam Tuliper	VSW12 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - Jeremy Clark	MAW03 Applied UX: iOS, Android, Windows - Anthony Handley	
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - Pacifica 7					
4:00 PM	5:15 PM	VSW13 Go Mobile with C#, Visual Studio, and Xamarin - James Montemagno	VSW14 To Be Announced	VSW15 Recruiters: The Good, The Bad, & The Ugly - Miguel Castro	VSW16 DI Why? Getting a Grip on Dependency Injection - Jeremy Clark	MAW04 Leveraging Azure Services - Kevin Ford	
8:00 PM	10:00 PM	Live! 360 Dessert Luau - Wantilan Pavilion - Sponsored by  amazon web services					

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Day 3: Thursday, November 19, 2015					
8:00 AM	9:15 AM	VSH01 Getting Started with ASP.NET 5 - <i>Scott Allen</i>	VSH02 Lessons Learned: Being Agile in a Waterfall World - <i>Philip Japikse</i>	VSH03 Windows, NUI and You - <i>Brian Randell</i>	VSH04 Improving Performance in .NET Applications - <i>Jason Bock</i>	MAH01 Building for the Modern Web with JavaScript Applications - <i>Allen Conway</i>	
9:30 AM	10:45 AM	VSH05 Build Data Driven Web Applications with ASP.NET MVC - <i>Rachel Appel</i>	VSH06 User Story Mapping - <i>Philip Japikse</i>	VSH07 Building Adaptive Uis for All Types of Windows - <i>Ben Dewey</i>	VSH08 Asynchronous Tips and Tricks - <i>Jason Bock</i>	MAH02 Building a Modern App with Xamarin - <i>Nick Landry</i>	
11:00 AM	12:15 PM	VSH09 Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI - <i>Marcel de Vries</i>	VSH10 Performance and Debugging with the Diagnostic Hub in Visual Studio - <i>Sasha Goldshtein</i>	VSH11 XAML Antipatterns - <i>Ben Dewey</i>	VSH12 Roslyn and .NET Code Gems - <i>Scott Allen</i>	MAH03 Building a Modern Cross-Platform App - <i>Brent Edwards</i>	
12:15 PM	1:30 PM	Lunch on the Lanai - <i>Lanai / Pacifica 7</i>					
1:30 PM	2:45 PM	VSH13 Hate JavaScript? Try TypeScript. - <i>Ben Hoelting</i>	VSH14 Advanced Modern App Architecture Concepts - <i>Marcel de Vries</i>	VSH15 WPF MVVM In Depth - <i>Brian Noyes</i>	VSH16 Getting More Out of Visual Studio Online: Integration and Extensibility - <i>Tiago Pascoal</i>	MAH04 DevOps And Modern Applications - <i>Dan Nordquist</i>	
3:00 PM	4:15 PM	VSH17 Grunt, Gulp, Yeoman and Other Tools for Modern Web Development - <i>Ben Hoelting</i>	VSH18 The Vector in Your CPU: Exploiting SIMD for Superscalar Performance - <i>Sasha Goldshtein</i>	VSH19 Building Maintainable and Extensible MVVM WPF Apps with Prism - <i>Brian Noyes</i>	VSH20 Readable Code - <i>John Papa</i>	MAH05 Analyzing Results with Power BI - <i>Steve Hughes</i>	
4:30 PM	5:45 PM	Live! 360 Conference Wrap-Up - <i>Andrew Brust (Moderator), Andrew Connell, Don Jones, Rockford Lhotka, Matthew McDermott, Brian Randell, & Greg Shields</i>					

START TIME	END TIME	Visual Studio Live! & Modern Apps Live! Post-Conference Workshops: Friday, November 20, 2015					
8:00 AM	5:00 PM	VSF01 Workshop: Angular in 0 to 60 - John Papa	VSF02 Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - Marcel de Vries & Roy Cornelissen			MAF01 Workshop: Modern App Development In-Depth - iOS, Android, Windows, and Web - Brent Edwards, Anthony Handley, & Allen Conway	

Sessions and speakers subject to change.

Build a Xamarin App with Authentication and Offline Support

Kraig Brockschmidt, Mike Wasson, Rick Anderson, Tom Dykstra, Erik Reitan

As described in the first part of this two-part series, “Create a Web Service with Azure Web Apps and WebJobs” (msdn.microsoft.com/magazine/mt185572), in the August issue, many mobile apps today are connected to one or more Web services that provide valuable and interesting data. Although it’s easy to just make direct REST API calls to those services and process responses in the client, such an approach can be costly in terms of battery power, bandwidth and throttling limitations imposed by different services. Performance might also suffer on lower-end hardware. It makes sense, then, to offload work to a custom back end, as we demonstrate with the “Altostratus” project we’ll discuss in this article.

This article discusses:

- Building a cross-platform mobile app using Xamarin.Forms
- Using social login to authenticate users with the application’s back end
- Using a SQLite database to maintain an offline cache of the back-end data in the mobile client
- Setting up a build environment for Xamarin apps using Visual Studio Online and Team Foundation Server

Technologies discussed:

Microsoft Azure, SQLite, ASP.NET, OAuth, Visual Studio, Visual Studio Online, Team Foundation Server, Xamarin, Xamarin.Forms

Code download available at:

aka.ms/altostratusproject

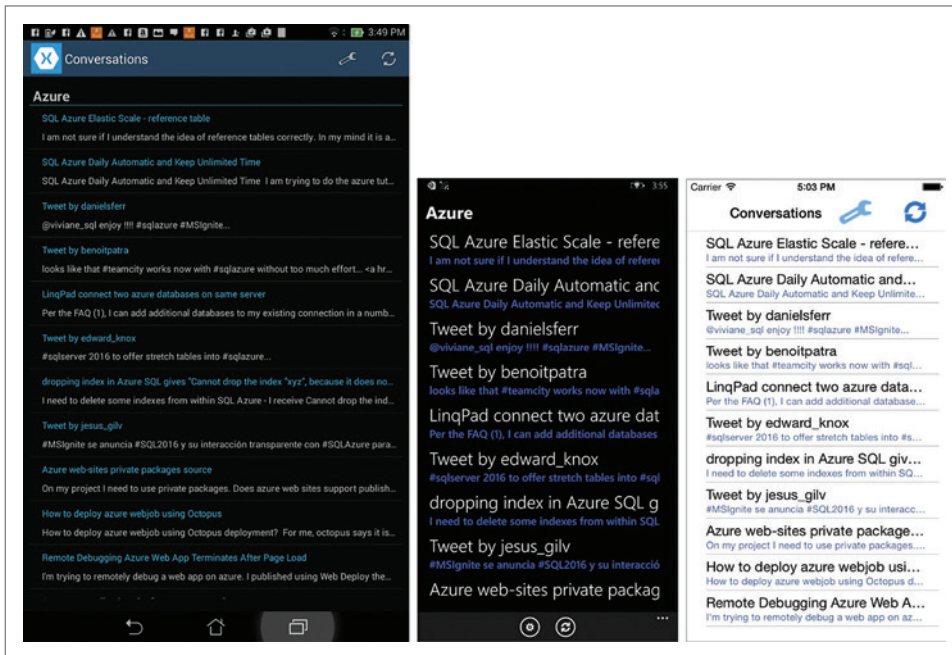
The Altostratus back end, discussed in Part 1, periodically gathers and normalizes “conversations” from StackOverflow and Twitter (just to use two different data sources) and stores them in a Microsoft Azure database. This means that the exact data needed by clients can be served up by the back end directly, allowing us to scale the back end in Azure to accommodate any number of clients without hitting the throttling limits of the original providers. By normalizing the data on the back end to match what the client needs, we also optimize data exchange through the Web API, conserving the ever-scarce resources of mobile devices.

In this article, we’ll discuss the details of the client app (see **Figure 1**). We’ll begin with the app’s architecture to set the overall context, then go into our use of Xamarin and Xamarin.Forms, authentication with the back end, building the offline cache, and building with Xamarin on Team Foundation Server (TFS) and Visual Studio Online.

Client App Architecture

The client app has three main pages or views: Configuration, Home and Item, whose classes share these names (see **Figure 2**). A secondary Login page has no UI and is simply the host for the OAuth provider Web pages. Using a basic Model-View-ViewModel (MVVM) structure, each main page except for Login has an associated view model class to handle the binding relationships between the views and the data model classes that represent items, categories and configuration settings (including a list of authentication providers).

(Note: Developers often prefer to separate the XAML views into a portable class library [PCL] that’s separate from the view models



Xamarin.Forms for a Cross-Platform Client

As you may have read in *MSDN Magazine*, Xamarin allows you to use C# and the Microsoft .NET Framework to build apps for Android, iOS and Windows, with a large amount of code that's shared among platforms. (For an overview of our development configuration, see the later section, "Building with Xamarin on TFS and VSO.") The Xamarin.Forms framework increases this amount further by providing a common UI solution for XAML/C#. With Xamarin.Forms, the Altostratus project shares more than 95 percent of its code in a single PCL. The only platform-specific code in the project, in fact, are the bits for startup that come from the project templates, renderers for the login pages that

and other classes, allowing designers to work independently with the views in tools like Blend. We didn't make that separation to keep the project structure simple, and also because Blend doesn't work with the controls in `Xamarin.Forms` at this time.)

The data model always populates these objects from a local SQLite database, which is pre-populated on first run of the app. Synchronization with the back end, which happens in the data model, is a straightforward process of retrieving new data (as little as possible to minimize network traffic), updating the database with that data, cleaning out any old data, and instructing the data model to refresh its objects. This triggers an update of the UI, thanks to data binding with the view models.

As we discuss later, a number of different events trigger a sync: a refresh button in the UI, changing the configuration, authenticating with the back end (which retrieves a previously saved configuration), resuming the app after at least 30 minutes, and so forth. Sync is the primary communication with the back end through its Web API, of course, but the back end also provides an API for registering a user, retrieving an authenticated user's settings, and updating those settings when an authenticated user changes the configuration.

deal with a Web browser control, and a few lines of code to copy the pre-populated SQLite database to the appropriate read-write location in local storage.

Note that a Xamarin project can also be configured to use a shared project rather than a PCL. However, Xamarin recommends using a PCL, and a primary benefit with Altostratus is we use the same PCL in a Win32 console application to create the

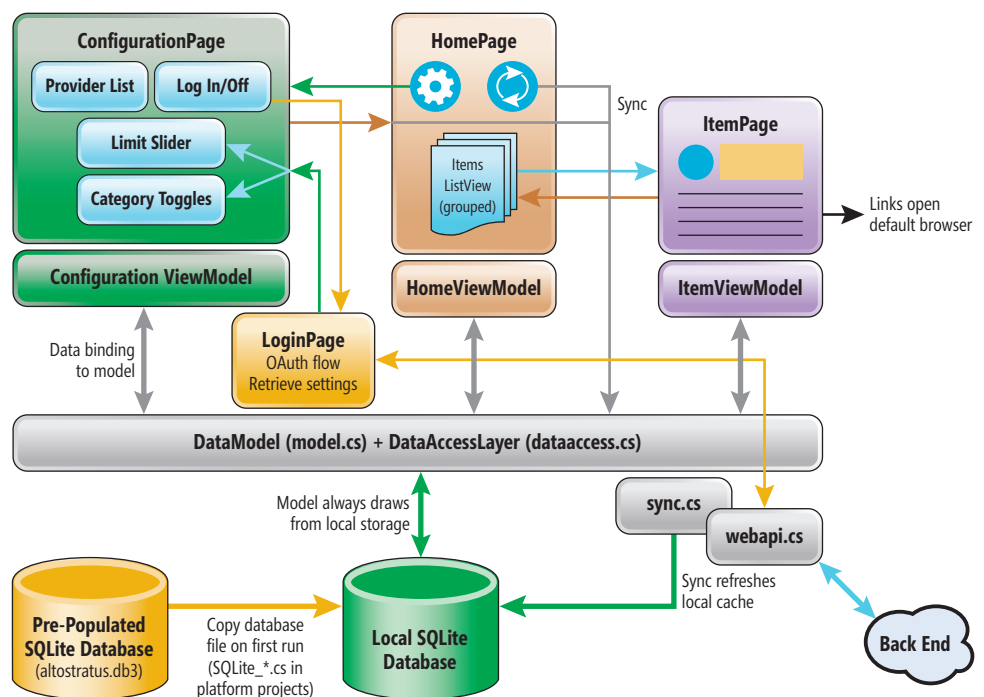


Figure 2 The Architecture of the Altostratus Client App, Showing the Names of the Primary Classes Involved (Files in the Project Match These Class Names Unless Indicated)

pre-populated database. This means we don't duplicate any database code for this purpose, and the initializer program will always be in sync with the rest of the app.

Be mindful that shared code doesn't much reduce the effort needed to thoroughly test the app on each target platform; that part of your process will take about as long as it would if you wrote each app natively. Also, because Xamarin.Forms is quite new, you may find platform-specific bugs or other behaviors that you'll need to handle in your code. For details on a few we found when writing Altostratus, see our post at bit.ly/1g5EF4j.

If you run into strange issues, your first stop should be the Xamarin bug database (bugzilla.xamarin.com). If you don't see your issue discussed there, post your question or issue on the Xamarin Forums (forums.xamarin.com), where you'll find the Xamarin staff to be quite responsive.

To work with any kind of user-specific data means authenticating that unique user.

That said, dealing with individual issues like these is far less work than learning the details of each individual platform's UI layer. And because Xamarin.Forms is relatively new, finding such issues helps the framework become increasingly robust.

Platform-Specific Adjustments

Within Xamarin.Forms, it's sometimes necessary to make adjustments for one platform or another, such as fine-tuning the layout. (Refer to Charles Petzold's excellent book, "Programming Mobile Apps with Xamarin.Forms" [bit.ly/1H8b2q6], for a number of examples.) You might also need to handle some behavioral inconsistencies, such as when a webview element fires its first Navigating event (again, for a few we encountered, see our post at bit.ly/1g5EF4j).

For this purpose, Xamarin.Forms has an API `Device.OnPlatform<T>(iOS_value, Android_value, Windows_value)`, and a matching XAML element. As you can guess, `OnPlatform` returns a different value depending on the current runtime. For example, the following XAML code hides the Configuration page's login controls on Windows Phone because the Xamarin.Auth component doesn't yet support that platform, so we always run unauthenticated (`configuration.xaml`):

```
<StackLayout Orientation="Vertical">
  <StackLayout.IsVisible>
    <OnPlatform x:TypeArguments="x:Boolean" Android="true" iOS="true"
      WinPhone="false" />
  </StackLayout.IsVisible>

  <Label Text="{ Binding AuthenticationMessage }" FontSize="Medium" />
  <Picker x:Name="providerPicker" Title="{ Binding ProviderListLabel }"
    IsVisible="{ Binding ProviderListVisible }" />
  <Button Text="{ Binding LoginButtonLabel}" Clicked="LoginTapped" />
</StackLayout>
```

Speaking of components, Xamarin itself is primarily built of components that abstract common capabilities of the native platforms, many of which predate Xamarin.Forms for UI. Some of these components are built into Xamarin, whereas others, including

community contributions, you acquire from components.xamarin.com. In addition to Xamarin.Auth, Altostratus uses the Connectivity Plugin (tinyurl.com/xconplugin) to show an indicator and disable the refresh button when the device is offline.

We found there's always some lag between when connectivity changes on the device (reflected in the plug-in's `IsConnected` property) and when the plug-in fires its event. This means there can be a few seconds between the device going offline and the Refresh button changing to the disabled state. To handle this, we use the Refresh command event to check the plug-in's `IsConnected` status. If it's offline, we immediately disable the button, but set a flag that tells the `ConnectivityChanged` handler to automatically start a sync when connectivity is restored.

Altostratus also uses Xamarin.Auth (tinyurl.com/xamauth) to handle details of authentication through OAuth, which we'll discuss in a moment. The caveat here is that the component presently supports only iOS and Android and not Windows Phone, and it wasn't in scope for our project to solve that particular shortcoming. Fortunately, the client app runs just fine unauthenticated, which simply means that the user's settings aren't maintained in the cloud and data exchange with the back end isn't fully optimized. When the component gets updated to support Windows, we need only remove the `OnPlatform` tag in the XAML shown earlier to make the login controls visible.

Authentication with the Back End

In the Altostratus application as a whole, we wanted to demonstrate the mechanics involved in storing some user-specific preferences on the back end such that the back end could automatically apply those preferences when handling HTTP requests. For this specific application, of course, we could have achieved the same result using just URI parameters with the requests, but such an example wouldn't serve as a basis for more complex scenarios. Storing preferences on the server also allows them to roam across all the user's devices.

To work with any kind of user-specific data means authenticating that unique user. This is different, mind you, from authorization. Authentication is a way to identify a user and validate they are who they claim to be. Authorization, on the other hand, relates to the permissions that any particular user has (for example, admin versus regular user versus guest).

For authentication, we use social login through third parties such as Google and Facebook, rather than implement our own credential system (and the back end has an API through which the client app retrieves a provider list for the Configuration page UI). The

Figure 3 Initializing the ASP.NET Identity Middleware

```
var fbOpts = new FacebookAuthenticationOptions()
{
    AppId = ConfigurationManager.AppSettings["FB_AppId"],
    AppSecret = ConfigurationManager.AppSettings["FB_AppSecret"]
};
fbOpts.Scope.Add("email");
app.UseFacebookAuthentication(fbOpts);

var googleOptions = new GoogleOAuth2AuthenticationOptions()
{
    ClientId = ConfigurationManager.AppSettings["GoogleClientId"],
    ClientSecret = ConfigurationManager.AppSettings["GoogleClientSecret"]
};
app.UseGoogleAuthentication(googleOptions);
```

Whitepaper Available
Four Ways to Optimize ASP.NET Performance

Extreme Performance Linear Scalability

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

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



Celebrating 10 Years of Market Leadership!



FREE Download

sales@alachisoft.com

US: +1 (925) 236 3830

www.alachisoft.com

Figure 4 Adding the Authorization Header

```
public interface ITokenProvider
{
    string AccessToken { get; }
}

class AuthenticationMessageHandler : DelegatingHandler
{
    ITokenProvider _provider;

    public AuthenticationMessageHandler(ITokenProvider provider)
    {
        _provider = provider;
    }

    protected override Task<HttpResponseMessage>
        SendAsync(HttpRequestMessage request,
            System.Threading.CancellationToken cancellationToken)
    {
        var token = _provider.AccessToken;
        if (!String.IsNullOrEmpty(token))
        {
            request.Headers.Authorization =
                new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", token);
        }
        return base.SendAsync(request, cancellationToken);
    }
}
```

primary advantage to social login is that we don't have to deal with credentials at all, or their attendant security and privacy concerns; the back end stores only an e-mail address as the user name, and the client manages only an access token at run time. Otherwise the provider does all the heavy lifting, including e-mail verification, password retrieval and so on.

Of course, not everyone has an account with a social login provider, and some users don't want to use social media accounts for privacy reasons. Also, social login may not be appropriate for line-of-business apps; for such cases we recommend Azure Active Directory. For our purposes, however, it's a logical choice because we simply need some means to authenticate an individual user.

Once authenticated, a user is authorized to save preferences on the back end. If we wanted to implement other levels of authorization (such as modifying other users' preferences), the back end could check the username against a permissions database.

Using OAuth2 for Social Login in ASP.NET Web API OAuth2 (bit.ly/1SxC1AM) is an authorization framework that allows users to grant access to resources, without sharing their credentials. It defines several "credential flows" that specify how credentials are passed around among various entities. ASP.NET Web API uses the so-called "implicit grant flow" in which the mobile app neither collects credentials nor stores any secrets. That work is done by the OAuth2 provider and the ASP.NET Identity library (asp.net/identity), respectively.

To enable social login, you must register your application with each of the login providers through their developer portals. (An "application" in this context means all client experiences, including mobile and Web, and isn't specifically tied to a mobile app). Once registered, the provider gives you a unique client ID and secret. See bit.ly/1BniZ89 for some examples.

We use these values to initialize the ASP.NET Identity middleware, as shown in **Figure 3**.

The strings like "FB_AppID" are keys that refer to the Web app's environment and config settings, where the actual IDs and secrets

are stored. This gives you the ability to update them without rebuilding and redeploying the app.

Using Xamarin.Auth to Handle Credential Flow Overall, the process of social authentication involves a variety of handshakes between the app, the back end and the provider. Fortunately, the Xamarin.Auth component (available in the Xamarin component store), handles most of this for the app. This includes working with the browser control and providing callback hooks so the app can respond when the authorization is complete.

Out of the box, Xamarin.Auth has the client app perform some parts of the authorization dance, which means the app stores the client ID and secret. That's slightly different from the ASP.NET flow, but Xamarin.Auth has a well-factored class hierarchy. The Xamarin.Auth.OAuth2Authenticator class derives from WebRedirectAuthenticator, which gives us the base functionality we need and requires us to write only a small amount of additional code, which is found in the LoginPageRenderer.cs files in the Android and iOS projects (because Xamarin.Auth doesn't yet support Windows). For more details on what we do here, see our blog post at tinyurl.com/kboathalto.

The client app, then, simply has a LoginPage class that derives from the base ContentPage of Xamarin.Forms, allowing us to navigate there. This class exposes two methods, CompleteLoginAsync, and CancelAsync, which are called from the LoginPageRenderer code depending on what the user does in the provider's Web interface.

Sending Authenticated Requests After a successful login, the client app has an access token. To make an authenticated request, it simply includes that token in an Authorization header like this:

```
GET http://hostname/api/UserPreferences HTTP/1.1
Authorization: Bearer I6zW8Dk...
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
```

Here, "Bearer" indicates authorization using bearer tokens, after which comes the long, opaque token string.

Overall, the process of social authentication involves a variety of handshakes between the app, the back end and the provider.

We use the System.Net.Http.HttpClient library for all REST requests with a custom message handler to add the authentication header to each request. Message handlers are plug-in components that allow you to examine and modify the HTTP request and response messages. To learn more, see bit.ly/1MyMMB8.

The message handler is implemented in the class AuthenticationMessageHandler (webapi.cs), and is installed when creating the HttpClient instance:

```
_httpClient = HttpClientFactory.Create(
    handler, new AuthenticationMessageHandler(provider));
```

The ITokenProvider interface is just a way for the handler to get the access token from the app (this is implemented in the UserPreferences class in model.cs). The SendAsync method is



2015 Orlando

ROYAL PACIFIC RESORT AT UNIVERSAL

November 16-20

The Ultimate Education Destination

Live! 360 is a unique conference where the IT and Developer community converge to test-drive leading edge technologies and fuel up on current ones. These five co-located events incorporate knowledge transfer and networking, along with finely tuned education and training, as you create your own custom conference, mixing and matching sessions and workshops to best suit your needs. All roads lead to Live! 360: **the ultimate education destination.**


VIEW ALL 180+ SESSIONS INSIDE
OPEN PAGE TO VIEW →


EVENT PARTNERS



PLATINUM SPONSORS



PLURALSIGHT

GOLD SPONSOR




SUPPORTED BY



PRODUCED BY



		VISUAL STUDIO LIVE! TRACKS						TECHMENTOR TRACKS						SQL SERVER LIVE! TRACKS							
		Cloud Computing	Database and Analytics	Lessons Learned and Advanced Practices	Mobile Client	Visual Studio / .NET Framework	Web Development	Windows Client	System Center and Config Management	Soft Skills for IT Pros	Security	Windows OS	Cloud & Mobile	Windows PowerShell	BI, Big Data, Data Analytics, and Data Visualization	SQL Server Administration & Maintenance	SQL Server Features & Components	SQL Server for Developers			
START TIME	END TIME	Visual Studio Live! Pre-Conference: Sunday, November 15, 2015						TechMentor Pre-Conference: Sunday, November 15, 2015						SQL Server Live! Pre-Conference: Sunday, November 15, 2015							
4:00 PM	9:00 PM	Pre-Conference Registration • Royal Pacific Resort Conference Center																			
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk																			
START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, November 16, 2015						TechMentor Pre-Conference Workshops: Monday, November 16, 2015						SQL Server Live! Pre-Conference Workshops: Monday, November 16, 2015							
8:00 AM	5:00 PM	VSM01 - Workshop: Service Oriented Technologies: Designing, Developing, & Implementing WCF and the Web API - Miguel Castro		VSM02 - Workshop: Triple D: Design, Development, and DevOps - Billy Hollis and Brian Randell		VSM03 - Workshop: Busy Developer's Guide to MEANJS - Ted Neward		TMM01 - Workshop: Windows PowerShell Scripting and Toolmaking - Jeff Hicks			TMM02 - Workshop: Service Management Automation for the Uninitiated - Jason Helmick			SQM01 - Workshop: SQL Internals, Optimization, and Indexing Deep Dive - Bradley Ball		SQM02 - Workshop: Big Data, NoSQL: Everything You Wanted to Learn - Andrew Brust					
5:00 PM	6:00 PM	EXPO Preview																			
6:00 PM	7:00 PM	LIVE! 360 KEYNOTE: Microsoft 3.0: New Strategy, New Relevance - Mary Jo Foley, Journalist and Author; with Andrew Brust, Senior Director, Datameer																			
START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, November 17, 2015						TechMentor Day 1: Tuesday, November 17, 2015						SQL Server Live! Day 1: Tuesday, November 17, 2015							
8:00 AM	9:00 AM	VISUAL STUDIO LIVE! KEYNOTE: The Future of Application Development - Visual Studio 2015 and .NET 2015 - Jay Schmelzer, Director of Program Management, Visual Studio Team, Microsoft						TECHMENTOR KEYNOTE: How to Get Your Company Moving Forward in Technology - Greg Shields, Author Evangelist, Pluralsight & Jason Helmick, Author, Pluralsight						SQL SERVER LIVE! KEYNOTE: Microsoft's Data Platform Landscape - Scott Klein, Corporate Technical Evangelist, Microsoft							
9:00 AM	9:30 AM	Networking Break • Visit the EXPO																			
9:30 AM	10:45 AM	VST01 - AngularJS 101 - Deborah Kurata		VST02 - A Tour of Azure for Developers - Adam Tuliper		VST03 - Busy Developer's Guide to NoSQL - Ted Neward		VST04 - Visual Studio, TFS, and VSO in 2015—What's New? - Brian Randell		TMT01 - Your Approach to BYOD Sucks! - Simon May		TMT02 - Deploying Windows Apps in the Cloud with RemoteApp - John O'Neill, Sr.		TMT03 - Secure Access from Everywhere! Implementing DirectAccess in Win2012R2 - Rich Hicks		SQT01 - Designing and Managing a SQL Server Replication Topology - Chad Churchwell		SQT02 - Encrypting Data in SQL Server - David Dye		SQT03 - Big Data Server - Andrew Brust	
11:00 AM	12:15 PM	VST05 - From ASP.NET Site to Mobile App in About an Hour - Ryan J. Salva		VST06 - Introduction to Next Generation of Azure PaaS - Service Fabric and Containers - Vishwas Lele		VST07 - Real World SQL Server Data Tools (SSDT) - Benjamin Day		VST08 - Automate Your Builds with Visual Studio Online or Team Foundation Server - Tiago Pascoal		TMT04 - Digging Deep into Microsoft InTune Patching and Mobile Management Solutions (BYOL - RA) - Mike Nelson		TMT05 - Integrating Active Directory with Azure, Step by Step - John O'Neill, Sr.		TMT06 - Using Sysinternals to Fight Malware - Michael Wiley		SQT04 - Hacking Expose: Using SSL to Secure SQL Server Connections - Chris Bell		SQT05 - SQL Server Reporting Services: Attendees Choose! - Kevin Gaff		SQT06 - Predictive Machine Learning - Andrew Brust	
12:15 PM	2:00 PM	Lunch • Visit the EXPO																			
2:00 PM	3:15 PM	VST09 - I Just Met You, and "This" is Crazy. But Here's My NaN, So Call(Me), Maybe? - Rachel Appel		VST10 - Cloud or Not, 10 Reasons Why You Must Know "Websites" - Vishwas Lele		VST11 - Windows 10 for Developers: What's New in Universal Apps - Nick Landry		VST12 - Defensive Coding Techniques in C# - Deborah Kurata		TMT07 - PowerShell Remoting Best Practices - Jeff Hicks		TMT08 - Why Your Company Needs a Cloud-Based Identity - Simon May		TMT09 - Windows 10—The Important New Features - Sami Laiho		SQT07 - Introduction to SQL Server Clustering - Chad Churchwell		SQT08 - Implementing Auditing in SQL Server - David Dye		SQT09 - Log Services 2015 - Andrew Brust	
3:15 PM	4:15 PM	Networking Break • Visit the EXPO																			
4:15 PM	5:30 PM	VST13 - Better Unit Tests through Design Patterns for ASP MVC, WebAPI, and AngularJS - Benjamin Day		VST14 - Running ASP.NET Cross Platform with Docker - Adam Tuliper		VST15 - Build Your First Mobile App in 1 Hour with Microsoft App Studio - Nick Landry		VST16 - To Be Announced		TMT10 - Mastering Regular Expressions in PowerShell - Jeff Hicks		TMT11 - Rocking Your Cloud with Docker! (BYOL / RA) - Mike Nelson		TMT12 - Windows Server 2016: Here's What We Know - Don Jones		SQT10 - The Spy Who Loathed Me: An Overview of SQL's Security Features - Chris Bell		SQT11 - Advanced SSIS Techniques - David Dye		SQT12 - Transition from Data Architect - Andrew Brust	
5:30 PM	7:30 PM	Exhibitor Reception																			
START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, November 18, 2015						TechMentor Day 2: Wednesday, November 18, 2015						SQL Server Live! Day 2: Wednesday, November 18, 2015							
8:00 AM	9:00 AM	LIVE! 360 KEYNOTE: DevOps: What it Means to You - Sponsored By: PLURALSIGHT - Don Jones, Curriculum Director for IT Pro Content, Pluralsight & Brian Randell, Partner, MCW Technologies																			
9:15 AM	10:30 AM	VSW01 - Mobile App Development with Xamarin and F# - Rachel Reese		VSW02 - Notify Your Millions of Users with Notification Hubs - Matt Milner		VSW03 - Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - Billy Hollis		VSW04 - To Git or Not to Git for Enterprise Development - Benjamin Day		TMW01 - Ethical Hacking: Methodologies and Fundamentals - Mike Danseglio & Avril Salter		TMW02 - Using Azure Site Recovery for Easy, Secure Off-Site Backup - John O'Neill, Sr.		TMW03 - Absolute Beginner's Guide to Advanced Hyper-V Networking - Greg Shields		SQW01 - SQL Server on Azure VMs: The Important Details - Thomas LaRock		SQW02 - Kick Start! SQL Server 2014 Performance Tips and Tricks - Pinal Dave		SQW03 - Implementing Patterns with T-SQL - Kevin Gaff	
10:30 AM	11:00 AM	Networking Break • Visit the EXPO																			
11:00 AM	12:15 PM	VSW05 - Automated UI Testing for Android and iOS Mobile Apps - James Montemagno		VSW06 - Busy Developer's Guide to the Clouds - Ted Neward		VSW07 - Designing and Building UX for Finding and Visualizing Data in XAML Applications - Billy Hollis		VSW08 - Anything C# Can Do, F# Can Do Better - Rachel Appel & Rachel Reese		TMW04 - Ethical Hacking: Recon, Data Gathering, and Target Selection - Mike Danseglio & Avril Salter		TMW05 - Real Time Monitoring of User Experience in VDI - Jeff Stokes		TMW06 - Sysinternals—The Latest and Greatest Updates! - Michael Wiley		SQW04 - SQL Server High Availability and Disaster Recovery - Chad Churchwell		SQW05 - Monitoring Databases in a Virtual Environment - Thomas LaRock		SQW06 - Getting the Most out of SQL Server - William D. Jones	
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO																			
1:45 PM	3:00 PM	VSW09 - Stop Creating Forms In Triplicate—Use Xamarin Forms - Matt Milner		VSW10 - To Be Announced		VSW11 - Developing Awesome 3D Games with Unity and C# - Adam Tuliper		VSW12 - Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - Jeremy Clark		TMW07 - Ethical Hacking: Destroying and Compromising Wireless Networks - Mike Danseglio & Avril Salter		TMW08 - Top Seven Tasks Required to Successfully Administer Office 365 - Justin Harris		TMW09 - Windows Internals Black Belt: Become a Troubleshooting Ninja - Sami Laiho		SQW07 - Just Enough SQL Administration to Keep it Running - Don Jones		SQW08 - Ancient Problems and Modern Solutions: Troubleshooting CPU in SQL - Pinal Dave		SQW09 - Introduction to Power BI - William D. Jones	
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m.																			
4:00 PM	5:15 PM	VSW13 - Go Mobile with C#, Visual Studio, and Xamarin - James Montemagno		VSW14 - To Be Announced		VSW15 - Recruiters: The Good, The Bad, & The Ugly - Miguel Castro		VSW16 - DI Why? Getting a Grip on Dependency Injection - Jeremy Clark		TMW10 - Ethical Hacking: Denial of Service and Social Engineering - Mike Danseglio & Avril Salter		TMW11 - Migrating to Office 365: Strategies - J. Peter Bruzzese		TMW12 - DISCUSSION: Career Strategies for the IT Pro - Greg Shields		SQW10 - To Be Announced		SQW11 - Configuring SQL Server for Performance... Like a Microsoft Certified Master - Thomas LaRock		SQW12 - Power BI and Analytics - William D. Jones	
8:00 PM	10:00 PM	Live! 360 Dessert Luau - Sponsored by 																			
START TIME	END TIME	Visual Studio Live! Day 3: Thursday, November 19, 2015						TechMentor Day 3: Thursday, November 19, 2015						SQL Server Live! Day 3: Thursday, November 19, 2015							
8:00 AM	9:15 AM	VSH01 - Getting Started with ASP.NET 5 - Scott Allen		VSH02 - Lessons Learned: Being Agile in a Waterfall World - Philip Japikse		VSH03 - Windows, NUI and You - Brian Randell		VSH04 - Improving Performance in .NET Applications - Jason Bock		TMH01 - Advanced PowerShell Traps and Gotchas - Don Jones		TMH02 - VDI Image Optimization and Troubleshooting - Jeff Stokes		TMH03 - Windows Internals Black Belt: Memory Management - Sami Laiho		SQH01 - Managing SQL Server at Scale - Joey D'Antoni		SQH02 - Indexes - the Unsung Heroes of SQL Server 2014 - Pinal Dave		SQH03 - Database Security - Leonard Label	
9:30 AM	10:45 AM	VSH05 - Build Data Driven Web Applications with ASP.NET MVC - Rachel Appel		VSH06 - User Story Mapping - Philip Japikse		VSH07 - Building Adaptive UIs for All Types of Windows - Ben Dewey		VSH08 - Asynchronous Tips and Tricks - Jason Bock		TMH04 - DSC Design and Planning - Don Jones		TMH05 - Securing Devices with InTune: Deep Dive - Justin Harris & Lawrence Novak		TMH06 - Building Better Hyper-V Guest Clusters - Bruce Mackenzie-Low		SQH04 - A Lap Around Multi-Site Clustering for SQL Server DBAs - Edwin Sarmiento		SQH05 - Dealing with Bad Roommates—SQL Server Resource Governor - Joey D'Antoni		SQH06 - Exploring Window Functions - Leonard Label	
11:00 AM	12:15 PM	VSH09 - Automated Cross Browser Testing of Your Web Applications with Visual Studio CodedUI - Marcel de Vries		VSH10 - Performance and Debugging with the Diagnostic Hub in Visual Studio - Sasha Goldshtein		VSH11 - XAML Antipatterns - Ben Dewey		VSH12 - Roslyn and .NET Code Gems - Scott Allen		TMH07 - Building a Secure DSC Pull Server - Jason Helmick		TMH08 - Managing and Securing Devices with Microsoft EMS - Justin Harris & Lawrence Novak		TMH09 - Windows Internals Black Belt: Security - Sami Laiho		SQH07 - Change Data Capture in SQL Server 2008/2012 - Kevin Gaff		SQH08 - Top 10 Wait Types Everyone Should Know - Janis Griffin		SQH09 - Python and SQL Server - Janis Griffin	
12:15 PM	1:30 PM	Lunch on the Lanai																			
1:30 PM	2:45 PM	VSH13 - Hate JavaScript? Try TypeScript - Ben Hoelzel		VSH14 - Advanced Modern App Architecture Concepts - Marcel de Vries		VSH15 - WPf MVVM In Depth - Brian Noyes		VSH16 - Getting More Out of Visual Studio Online: Integration and Extensibility - Tiago Pascoal		TMH10 - Creating Class-Based DSC Resources - Jason Helmick		TMH11 - Exchange and O365 PowerShell - Michael O'Neill		TMH12 - DISCUSSION: Delivering Better Presentations - Don Jones		SQH10 - Geekin' Out on Extended Events - The System Health Session - Janis Griffin		SQH11 - Troubleshooting SQL Server Performance using Wait Statistics - Edwin Sarmiento		SQH12 - Business Intelligence with DataViz Tool - Janis Griffin	
3:00 PM	4:15 PM	VSH17 - Grunt, Gulp, Yeoman and Other Tools for Modern Web Development - Ben Hoelzel		VSH18 - The Vector in Your CPU: Exploiting SIMD for Superscalar Performance - Sasha Goldshtein		VSH19 - Building Maintainable and Extensible MVVM WPf Apps with Prism - Brian Noyes		VSH20 - Readable Code - John Papa		TMH13 - PowerShell Unplugged: Stuff I Stole from Snover - Don Jones		TMH14 - O365 and Exchange Hybrid Management - Michael O'Neill		TMH15 - Windows 10 Deployment with Config Man OSD—Planning and Strategy - Steven Rachui		SQH13 - Building Perfect SQL Servers, Every Time - Joey D'Antoni		SQH14 - Proactively Identifying and Dealing with SQL Server Database Corruption - Edwin Sarmiento		SQH15 - Eye Vegetables to Visualize - Janis Griffin	
4:30 PM	5:45 PM	LIVE! 360 CONFERENCE WRAP-UP - Andrew Brust (Moderator), Andrew Connell, Don Jones, Rockford Lhotka, Matthew McDermott, Brian Randell, & Greg Shields																			
START TIME	END TIME	Visual Studio Live! Post-Conference Workshops: Friday, November 20, 2015						TechMentor Post-Conference Workshops: Friday, November 20, 2015						SQL Server Live! Post-Conference Workshops: Friday, November 20, 2015							
8:00 AM	5:00 PM	VSF01 - Workshop: Angular in 0 to 60 - John Papa				VSF02 - Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - Marcel de Vries & Roy Cornelissen		TMF01 - Workshop: Mastering Windows Troubleshooting: BSODs, System Hangs, and Performance Problems (BYOL) - Bruce Mackenzie-Low			TMF02 - Workshop: Application Deployment—The Configuration Manager Way - Steven Rachui			SQF01 - Workshop: Windows Server Failover Clustering for the SQL Server DBA - Edwin Sarmiento		SQF02 - Workshop: SQL Server for Developers - Leonard Label					

November 16 - 20
ROYAL PACIFIC RESORT AT UNIVERSAL

2015
Orlando

SharePoint LIVE!
TRAINING FOR COLLABORATION

ModernApps LIVE!
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

	SHAREPOINT LIVE! TRACKS						MODERN APPS LIVE! TRACK	
SQL Server Performance Tuning and Optimization	Developing Add-ins and Solutions for SharePoint	High-Value SharePoint Workloads: Social, Search, BI, and Business Process Automation	Information and Content Management: Documents, Records, and Web	Keys to SharePoint Success: Strategy, Governance, Adoption	SharePoint Infrastructure Management and Administration	SharePoint, Office 365 and the Cloud	Presented in Partnership with:	

SharePoint Live! Pre-Conference: Sunday, November 15, 2015						MAL! Pre-Conf.: Sun., Nov. 15

6, 2015	SharePoint Live! Pre-Conference Workshops: Monday, November 16, 2015		MAL! Pre-Conf. Wrkshp: Mon., Nov. 16
Analytics and But Were Afraid to Ask -	SPM01 - Workshop: Developer OnRamp to the Office 365 & Azure AD Highway! - Andrew Connell	SPM02 - Workshop: Getting up and Running with Office 365 for IT Pros - Dan Usher & Scott Haag	MAM01 - Workshop: Modern App Technology Overview — Android, iOS, Cloud, and Mobile Web - Nick Landry, Kevin Ford, & Steve Hughes

SharePoint Live! Day 1: Tuesday, November 17, 2015						MAL! Day 1: Tues., Nov. 17
SHAREPOINT LIVE! KEYNOTE: Conference Welcome & State of SharePoint & Office 365 - Andrew Connell, Independent Contractor, Andrew Connell Inc. & Matt McDermott, Founder and Director, Aptillon, Inc.						MODERN APPS LIVE! KEYNOTE: The Future of Application Development - Visual Studio 2015 and .NET 2015 - Jay Schmelzer, Director of Program Management, Visual Studio Team, Microsoft

ig Data with SQL and Hadoop Andrew Brust	SPT01 - Integrating Office Web Apps with SharePoint 2013 - Brian Alderman	SPT02 - An Introduction to SharePoint 2013 Add-ins for Developers - Rob Windsor	SPT03 - 10 Things You Should Know About Office 365 - Martina Grom	MAT01 - Defining Modern App Development - Rockford Lhotka
ive Analytics and Azure ne Learning Andrew Brust	SPT04 - Succeeding with Hybrid SharePoint and Search: Strategy and Implementation Including SharePoint 2016 - Jeff Fried	SPT05 - SharePoint Developers, Come to the Client Side (We Have Cookies) - Bill Ayers	SPT06 - No Governance as Usual - Robert Bogue	MAT02 - Modern App Architecture - Brent Edwards
Using Integration 014 (SSIS) to Load use - Thomas LeBlanc	SPT07 - To Be Announced	SPT08 - To Be Announced	SPT09 - Intro to Content Web Parts: the Sweetest Way to View and Manage Content - Kim Frehe	MAT03 - ALM with Visual Studio Online (TFS) and Git - Brian Randell
Skills Needed to DBA to Data Warehouse - Thomas LeBlanc	SPT10 - Configuring SharePoint Hybrid Search - Matthew McDermott	SPT11 - Introduction to Node.js for the SharePoint & Office 365 Developer - Andrew Connell	SPT12 - OneDrive for Business: Much More than a File Share - Erica Tolle	MAT04 - Reusing Logic Across Platforms - Kevin Ford

	SharePoint Live! Day 2: Wednesday, November 18, 2015			MAL! Day 2: Wed., Nov. 18
enting Data Warehouse he Microsoft BI Tools evin Goff	SPW01 - SQL 2014 Always On Availability Groups for SharePoint 2013 - Vlad Catrinescu	SPW02 - SharePoint Development Lifecycle for Solutions and Add-ins - Robert Bogue	SPW03 - SharePoint 2013 Workflow Basics in Plain English for Business Users - Kim Frehe	MAW01 - Coding for Quality and Maintainability - Jason Bock
g Started with Analysis 2012 Tabular n E. Pearson III	SPW04 - Automating SharePoint On- Premises Tasks with PowerShell - Ben Stegink	SPW05 - Building Office Add-Ins with Visual Studio - Bill Ayers	SPW06 - Understanding Your SharePoint 2013 Business Intelligence Options - Darren Bishop	MAW02 - Start Thinking Like a Designer - Anthony Handley
ing Forensic Analytics with ivot for Excel n E. Pearson III	SPW07 - Managing Data Recovery in SharePoint 2013 - Brian Alderman	SPW08 - Building Search-Driven Apps with the Office Graph - Jeff Fried	SPW09 - Plan to Migrate to SharePoint Online - Erica Tolle	MAW03 - Applied UX: iOS, Android, Windows - Anthony Handley
20: Analytics in the Cloud d in Excel Andrew Brust	SPW10 - Setting Up Directory Synchronization for Office 365 - Dan Usher & Scott Haag	SPW11 - Utilizing jQuery in SharePoint - Get More Done Faster - Mark Rackley	SPW12 - Building an Award-Winning Wellness Program Using SharePoint - Sandra Mahan	MAW04 - Leveraging Azure Services - Kevin Ford

	SharePoint Live! Day 3: Thursday, November 19, 2015			MAL! Day 3: Thurs., Nov. 19
Development with SQL Data Tools Standard Label	SPH01 - To the Cloud! Using IaaS as a Hosting Provider for SharePoint - <i>Dan Usher & Scott Haag</i>	SPH02 - Using jQuery to Maximize SharePoint Form Usability - <i>Mark Rackley</i>	SPH03 - Three Must-Have SharePoint Workflows with SharePoint Designer - <i>Vlad Catrinescu</i>	MAH01 - Building for the Modern Web with JavaScript Applications - <i>Allen Conway</i>
ing T-SQL Enhancements: Using and More Standard Label	SPH04 - Getting Started with PowerShell for Office 365 - <i>Ben Stegink</i>	SPH05 - Building Secure Client Applications for Office 365 - <i>Bill Ayers</i>	SPH06 - Content Management Using the Power of Search - <i>Kim Frehe</i>	MAH02 - Building a Modern App with Xamarin - <i>Nick Landry</i>
nd R for SQL and Business ce Professionals en Strupp	SPH07 - Building Great Search Experiences in SharePoint - <i>Matthew McDermott</i>	SPH08 - Using Azure to Replace Server-Side Code in Office 365 - <i>Paul Schaefflin</i>	SPH09 - To Be Announced	MAH03 - Building a Modern Cross-Platform App - <i>Brent Edwards</i>
Intelligence Barista: What to Use, and When? en Strupp	SPH10 - Information Management Strategy with Office 365 in Mind - <i>Jeff Fried</i>	SPH11 - Introduction to the SharePoint Client Object Model and REST API - <i>Rob Windsor</i>	SPH12 - Working Together with Groups, Yammer and Exchange Online - <i>Martina Grom</i>	MAH04 - DevOps And Modern Applications - <i>Dan Nordquist</i>
ables and Eye Candy: How ilize Your Data en Strupp	SPH13 - To Be Announced	SPH14 - Mixing in Code with a No-Code Access Services Applications to Get the Job Done! - <i>Darrin Bishop</i>	SPH15 - How to Fire People Using SharePoint - <i>Sandra Mahan</i>	MAH05 - Analyzing Results with Power BI - <i>Steve Hughes</i>

SharePoint Live! Post-Conference Workshops: Friday, November 20, 2015						MAL! Post-Conf. Wrkshp: Fri., Nov. 20
ver 2014	SPF01 - Workshop: Deep Dive into Authentication and Authorization in the Cloud - Paul Schaefflin		SPF02 - Workshop: Create Your End User Adoption Strategy - Erica Tolle			MAF01 - Workshop: Modern App Development In-Depth— iOS, Android, Windows, and Web - Brent Edwards, Anthony Handley, & Allen Conway

REGISTER NOW AT **LIVE360EVENTS.COM**

5
Great
Conferences
1
Great Price



Connect with Live! 360



twitter.com/live360
@live360



facebook.com
Search "Live 360"



linkedin.com
Join the "Live! 360" group!

REGISTER BY SEPTEMBER
16 AND **SAVE \$400!**



Use promo code
L360SEPT1

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

Take the Tour

Five events, 25+ tracks, and literally 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 LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

CODE IN THE SUN and experience unbiased and practical development training on the Microsoft Platform.

SharePoint LIVE!

TRAINING FOR COLLABORATION

Chart the best COLLABORATION COURSE with SharePoint Live!, providing leading-edge SharePoint knowledge and training

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

SQL Server Live! will leave you with the skills needed to **DRIVE DATA FORWARD**, whether you are a DBA, developer, IT Pro, or Analyst.

ModernApps LIVE!

MOBILE CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with Magenit, this unique conference **LEADS THE WAY** to learning how to architect, design and build a complete Modern App.

TECHMENTOR

This is IT TRAINING THAT'S WORTH THE TRAVEL, focused entirely on making your datacenter more modern, more capable, and more manageable.

LIVE360EVENTS.COM



FEATURED SPEAKERS

Live! 360 brings you over 70 speakers; some of the best and brightest experts in the industry.

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS



RACHEL APPEL



JASON BOCK



BILLY HOLLIS



MIGUEL CASTRO



MARCEL DE VRIES



DEBORAH KURATA



NICK LANDRY



BRIAN NOYES



JOHN PAPA



BRIAN RANDELL

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS



JEFFERY HICKS



DON JONES



SAMI LAIHO



BRUCE MACKENZIE-LOW



GREG SHIELDS

SharePoint LIVE!

TRAINING FOR COLLABORATION



BILL AYERS



ROBERT BOGUE



ANDREW CONNELL



MATTHEW MCDERMOTT



SANDRA MAHAN

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS



ANDREW BRUST



PINAL DAVE



EDWIN SARMIENTO



LEONARD LOBEL



TED NEWARD

ModernApps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT



ALLEN CONWAY



BRENT EDWARDS



KEVIN FORD



STEVE HUGHES



ROCKFORD LHOTKA

REGISTER BY SEPTEMBER
16 AND **SAVE \$400!**

Use promo code L360SEPT1



CONNECT WITH LIVE! 360

twitter.com/@live360events
facebook.com - Search "Live 360"
linkedin.com - Join the "Live 360" group!

VIEW THE FULL LIST OF SPEAKERS AT LIVE360EVENTS.COM

Azure Offline Sync

An alternative to implementing your own offline cache is Azure Offline Sync for tables, which is part of Azure Mobile Services. This eliminates the need to write any sync code at all, and works for pushing changes from the client to the server. Because it uses table storage, however, it doesn't provide a relational data model as SQLite does.

called for every HTTP request; as **Figure 4** shows, it will add the Authorization header if the token provider has one to use.

On the back end, as described in Part 1 of this article, if a token is received with a request, the token is used to retrieve that user's preferences and apply them automatically to the rest of the request. For example, if the user has set the conversation limit to 25 instead of the default of 100, then a maximum of 25 items will be returned with a request, saving network bandwidth.

Building an Offline Cache for Back-End Data

A big advantage of a mobile app over the mobile Web is the flexibility to support offline usage. A mobile app, by definition, is always present on the user's device and can use a variety of data storage options, such as SQLite, to maintain an offline data cache, rather than relying on browser-based mechanisms.

The UI of the Altostratus mobile client, in fact, simply works with data that's maintained in a local SQLite database, making the app fully functional without connectivity. When connectivity is present, background processes retrieve current data from the back end to update the database. This updates the data model objects that sit above the database, which in turn triggers UI updates through data binding (you can see this back in **Figure 2**). In this way it's a very similar architecture to the back end discussed in Part 1, where the ongoing webjobs collect, normalize and store data in a SQL Server database so that the Web API can service requests directly from that database.

Offline support for Altostratus involves three distinct tasks:

- Placing a pre-populated database file directly in the app package to provide working data immediately on first run without requiring connectivity.
- Implementing one-way synchronization processes for each part of the data model that gets surfaced in the UI: conversations (items), categories, authentication providers and user preferences.

Figure 5 Retrieving the Platform-Specific Location of the Database File

```
public DataAccessLayer(SQLiteAsyncConnection db = null)
{
    if (db == null)
    {
        String path = DependencyService.Get<ISQLite>().GetDatabasePath();
        db = new SQLiteAsyncConnection(path);

        // Alternate use to use the synchronous SQLite API:
        // database = SQLiteConnection(path);
    }

    database = db;
    _current = this;
}
```

- Hooking up synchronization to appropriate triggers beyond the Refresh button in the UI.

Let's look at each of these in turn.

Creating a Prepopulated Database It's possible for a user to install an app from an online store but not run it until a later time when the device is offline. The question is, do you want your app to say, "Sorry, we can't do anything useful unless you're online?" Or do you want your app to handle such cases intelligently?

To demonstrate the latter approach, the Altostratus client includes a prepopulated SQLite database directly in its app package on each platform (located in each platform project's resources/raw folder). On first run, the client copies this database file to a read-write location on the device and then works with it from there exclusively. Because the file copy process is unique to each platform, we use the Xamarin.Forms.DependencyService feature to resolve, at runtime, a specific implementation of an interface we define called ISQLite. This happens from the DataAccessLayer constructor (DataAccess.cs), which then calls ISQLite.GetDatabasePath to retrieve the platform-specific location of the copied read-write database file, as shown in **Figure 5**.

A big advantage of
a mobile app over the mobile
Web is the flexibility to support
offline usage.

To create the initial database, the Altostratus solution contains a small Win32 console application called DBInitialize. It uses the same shared PCL as the app to work with the database, so there's never an issue of having a second, mismatched code base. DBInitialize, however, doesn't need to use the DependencyService: It can just create a file directly and open a connection:

```
string path = "Altostratus.db3";
SQLiteAsyncConnection conn = new SQLiteAsyncConnection(path);
var dbInit = new DataAccessLayer(conn);
```

From here, DBInitialize calls DataAccessLayer.InitAsync to create the tables (something the app never has to do with the prepopulated database), and uses the other DataAccessLayer methods to get data from the back end. Note that with async calls, DBInitialize just uses .Wait; because it's a console application and doesn't need to worry about responsive UI:

```
DataModel model = new DataModel(dbInit);
model.InitAsync().Wait();
model.SyncCategories().Wait();
model.SyncAuthProviders().Wait();
model.SyncItems().Wait();
```

To give users something to look at, it uses a timer to put a dot on the screen every half-second.

Note that you'll always want to check the prepopulated database with a tool like DB Browser for SQLite (bit.ly/10Ckm8Y) before pulling it into the project. It's possible for one or more of the Web requests to fail, in which case the database wouldn't be valid. You could build this logic into DBInitialize so it deletes the database

and shows an error. In our case, we just watch for error messages and run the program again, if needed.

You might be asking, “Won’t the contents of a prepopulated database go out of date relatively quickly? I wouldn’t want the users of my app to see really stale data on first run!” This will, of course, be the case if you don’t regularly update your app. So you’ll want to commit to periodic app updates that include a database with reasonably current data (depending on the nature of your data).

If the user already has the app installed, the update won’t have any affect because the code that copies the packaged database file checks whether a read-write copy already exists, and just uses that. However, if file existence was the only such check, a user who hasn’t run the app for a while could end up launching the app with older data than what’s actually in a more recently updated package. You could check whether the timestamp of the cache database is older than the in-package one, and overwrite the cache with the newer copy. This isn’t something we implemented in Altostratus, however, because we’d also need to preserve the user preferences information from the existing database.

Synchronizing the Offline Cache with the Back End As noted before, the Altostratus client always runs against its cache database. All Web requests to the back end (except for uploading new user preferences) happen in the context of synchronizing the cache. The mechanism for this is implemented as part of the `DataModel` class, specifically through the four methods in `sync.cs`: `SyncSettings` (which delegates to `Configuration.ApplyBack end-Configuration` in `model.cs`), `SyncCategories`, `SyncAuthProviders` and `SyncItems`. Clearly the workhorse of the bunch is `SyncItems`, but we’ll talk about what triggers all of these in the next section.

Note that Altostratus synchronizes data in one direction only, from the back end to the local cache. Also, because we know that the data we’re interested in doesn’t change all that rapidly (given the

schedule for the back-end webjobs), we care only about eventual consistency with the back-end data store rather than updating on the order of minutes or seconds.

Each sync process is essentially the same: Retrieve the current data from the back end, update the local database with new values, remove any old values from the database and then repopulate the data model objects to trigger UI updates.

For items, there’s a little more work because `SyncItems` can be invoked from the UI, and we want to guard against over-excited users who press the button repeatedly. The private `DataModel.syncTask` property indicates whether there’s an active item sync; `SyncItems` ignores a repeat request if `syncTask` is non-null. Furthermore, because an item request might take a while and involves larger data sets, we want to be able to cancel an item sync if the device goes offline. To do so, we save a `System.Threading.CancellationToken` for the task.

The private `SyncItemsCore` method, shown in **Figure 6**, is the heart of the process. It retrieves the timestamp of the last sync from the database and includes that with the Web request.

By doing this, the back end returns only those items that are new or updated since the given time. As a result, the client gets only the data it really needs, conserving the user’s potentially limited data plan. This also means the client does less work to process data from each request, which also conserves battery power and minimizes network traffic. Though it doesn’t sound like much, handling five conversations per category instead of, say, 50, with each request is a 90 percent reduction. At 20 to 30 syncs per day, this could easily add up to hundreds of megabytes in a month for just the one app. In short, your customers will certainly appreciate the efforts you make to optimize the traffic!

Once the request comes back, the `ProcessItems` method adds all those items to the database, doing a little cleanup of titles (such as removing smart quotes) and extracting the first 100 characters of the body for a description to show in the main list. The title cleanup is something we could have the back end do instead, which might save just a little processing time on the client. We’ve chosen to leave it on the client because other scenarios might need to do platform-specific adjustments. We could also have the back end create the 100-character descriptions, which would mean saving a little client processing but increasing the network traffic. With the data we’re working with here, it’s probably an even trade-off, and because the UI is ultimately the client’s responsibility, it seems better to leave the client in control of this step. (For additional details on this and a couple other UI considerations, see our blog post tinyurl.com/kboathaltotextra.)

Once we’ve added items to the database, the data model groups are refreshed through `PopulateGroupedItemsFromDB`. Here it’s important to understand that the database probably has more items than necessary for the user’s current conversation-limit setting. `PopulateGroupedItemsFromDB` accounts for this by applying that limit directly to the database query.

Over time, though, we don’t want the database to keep expanding by retaining a bunch of items that will never be displayed again. For this, `SyncItems` calls a method `DataAccessLayer.ApplyConversationLimit` to cull old items from the database until the number of items matches a specified limit. Because the size of any individual item within the Altostratus data set is relatively small, we use the maximum conversation limit of 100 regardless of the user’s current setting.

Figure 6 The Private `SyncItemsCore` Method

```
private async Task<SyncResult> SyncItemsCore()
{
    SyncResult result = SyncResult.Success;
    HttpResponseMessage response;

    Timestamp t = await DataAccessLayer.Current.GetTimestampAsync();
    String newRequestTimestamp =
        DateTime.UtcNow.ToString(WebAPIConstants.ItemsFeedTimestampFormat);

    response = await WebAPI.GetItems(t, syncToken);

    if (!response.IsSuccessStatusCode)
    {
        return SyncResult.Failed;
    }

    t = new Timestamp() { Stamp = newRequestTimestamp };
    await DataAccessLayer.Current.SetTimestampAsync(t);

    if (response.StatusCode == System.Net.HttpStatusCode.NoContent)
    {
        return SyncResult.NoContent;
    }

    var items = await response.Content.ReadAsAsync<IEnumerable<FeedItem>>();
    await ProcessItems(items);

    // Sync is done, refresh the ListView data source.
    await PopulateGroupedItemsFromDB();

    return result;
}
```



Create & Edit PDF's in .Net - ActiveX - WinRT - Universal Apps

NEW
v5.5

- 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 Universal Apps DLLs enable publishing C#, C++CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module



Complete Suite of Accurate PDF Components

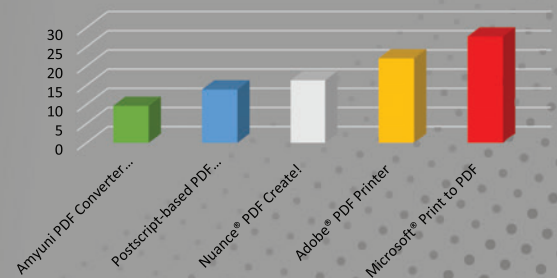
- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML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



High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



Other Developer Components from Amyuni

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need



AMYUNI 
Technologies

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 development tools available at
www.amyuni.com

This way, if the user raises that limit, we don't have to request data again from the back end. If we had much larger data items, however, it might make more sense to aggressively cull the database and request items again if they're needed.

Synchronization Triggers The Refresh button in the UI is clearly the primary way an item sync happens, but when do the other sync processes take place? And are there other triggers for an item sync?

It's a bit tricky to answer these questions with regard to the code, because all calls to the Sync* methods happen in one place, the HomeViewModel.Sync method. However, this method, and a secondary entry point, HomeViewModel.CheckTimeAndSync, are called from a variety of other places. Here's a summary of when, where and how calls to Sync are parameterized with a value from the SyncExtent enumeration:

- On startup, the HomeViewModel constructor calls Sync(SyncExtent.All), using a fire-and-forget pattern so the sync happens entirely in the background. The pattern here simply means saving the return value from an async method into a local variable to suppress a compiler warning about not using await.
- Inside the handler for the Connectivity plug-in's ConnectivityChanged event, we call Sync if the device was offline when a previous call was made (using the same extent as requested then).
- If the user visits the Configuration page and makes changes to the active categories or the conversation limit, or logs into the back end and, thus, applies settings from the back end, that fact is remembered by the DataModel.Configuration.HasChanged flag. When the user returns to the homepage, the HomePage.OnAppearing handler calls HomeViewModel.CheckRefresh, which checks HasChanged and calls Sync(SyncExtent.Items), if needed.

- The App.OnResume event (app.cs) calls CheckTimeAndSync, which applies some logic to determine what should be synced based on how long the app has been suspended. Clearly, these conditions are highly dependent on the nature of your data and back-end operations.
- Finally, the Refresh button calls CheckTimeAndSync with a flag to always do at least an item sync. The Refresh button uses CheckTimeAndSync because it's possible—albeit quite rare—that a user might have left the app running in the foreground for more than half an hour or even a day, in which case the Refresh button should also do the other syncs as we do when resuming.

A benefit of consolidating everything into HomeViewModel.Sync is that it can set the public HomeViewModel.IsSyncing property at the appropriate times. This property is data-bound to both the IsVisible and IsRunning properties of a Xamarin.Forms.ActivityIndicator in Home.xaml. The simple act of setting this flag controls the visibility of that indicator.

Building with Xamarin on TFS and Visual Studio Online

For the Altostratus project, we used a development environment that's somewhat common for cross-platform work: a Windows PC with emulators and tethered devices for Android and Windows Phone, along with a local Mac OS X machine with the iOS simulator and tethered iOS devices (see **Figure 7**). With such a setup, you can do all development and debugging work directly within Visual Studio on the PC, using the Mac OS X machine for remote iOS builds and debugging. Store-ready iOS apps can then also be submitted from the Mac.

We employed Visual Studio Online for team collaboration and source control, and configured it to do continuous integration builds for both the back end and the Xamarin client. Had we started this project today, we'd be able to use the latest Visual Studio Online build system to build Xamarin apps directly in the hosted build controller. You can find out more from our blog post at tinyurl.com/kboauthxamvso. Earlier in 2015, however, the hosted build controller didn't yet have this support. Fortunately, it's a simple matter—honestly!—to use a local machine running TFS as the build controller for Visual Studio Online. On that server we installed the free TFS Express edition along with Xamarin and the necessary Android and Windows platform SDKs, making sure to place the Android SDK in a location such as c:\android-sdk, which the build account could access. (Its installer by default places the SDK in the current user's storage, to which the build account doesn't have permissions.) This is discussed in the Xamarin documentation, “Configuring Team Foundation Server for Xamarin,” at bit.ly/10hQPSW.

Once the build server is fully configured, the following steps make the connection to Visual Studio Online (see “Deploy and Configure a Build Server” at bit.ly/1RJS4QL):

1. Open the TFS Administration Console.
2. In the left-hand navigation pane, expand the server name and select Build Configuration.

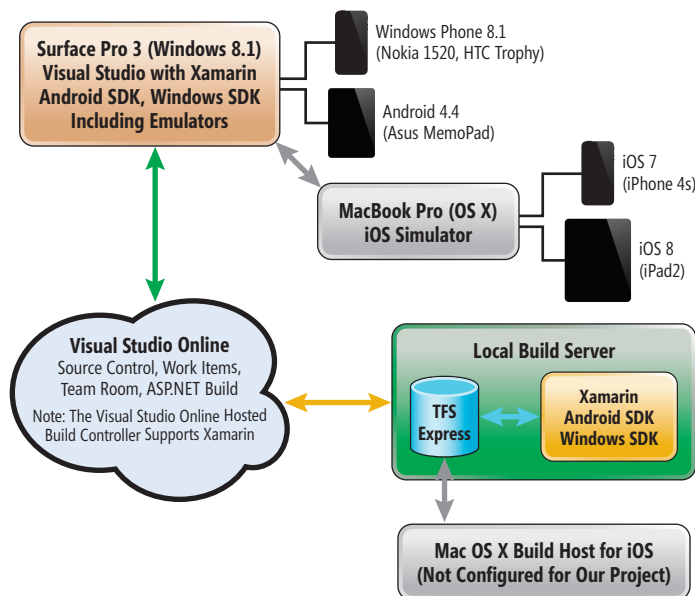


Figure 7 A Common Cross-Platform Development Environment for Xamarin Projects, As Well As Those That Use Other Technologies Like Visual Studio Tools for Apache Cordova

3. Under the build service, click Properties to open the Build Service Properties dialog.
4. Click "Stop the service" at the top of the dialog.
5. Under Communications, in the box under Provide Build Services for Project Collection, enter the URL of your Visual Studio Online collection, such as `https://<your account>.visualstudio.com/defaultcollection`.
6. Click the Start button at the bottom of the dialog to restart the service.

That's all it takes! When you create a build definition in Visual Studio Team Explorer, the TFS machine connected to Visual Studio Online will appear in the list of available build controllers. By selecting that option, builds that you queue from Visual Studio or that are queued upon check-in will be routed to your TFS machine.

Wrapping Up

We hope you've enjoyed our discussion on the Altostratus project, and that you'll find the code useful for your own mobile, cloud-connected apps. Our goal with this project, again, was to provide a clear example of a cross-platform mobile app with a custom back end that could do significant work on behalf of the client to optimize the work that had to be done on the client directly. By having the always-running back end collect data on behalf of all clients, we greatly reduced the amount of client-generated network traffic (and its subsequent impact on data plans). By normalizing the data from different sources, we minimized the amount of data processing necessary on the client, which conserves ever-important battery power. And by authenticating a user with the back end, we demonstrated how it's possible to store user preferences there and have them automatically applied to the client's interactions with the back end, again optimizing network traffic and processing requirements. We understood that for our specific requirements there might have been easier ways to achieve the same effect, but we wanted to create an example that would be scalable to more complex scenarios.

We'd love to hear what you think about this project. Let us know! ■

KRAIG BROCKSCHMIDT works as a senior content developer for Microsoft and is focused on cross-platform mobile apps. He's the author of "Programming Windows Store Apps with HTML, CSS and JavaScript" (two editions) from Microsoft Press and blogs on kraigbrockschmidt.com.

MIKE WASSON is a content developer at Microsoft. For many years he documented the Win32 multimedia APIs. He currently writes about Microsoft Azure and ASP.NET.

RICK ANDERSON works as a senior programming writer for Microsoft, focusing on ASP.NET MVC, Microsoft Azure and Entity Framework. You can follow him on Twitter at twitter.com/RickAndMSFT.

ERIK REITAN is a senior content developer at Microsoft. He focuses on Microsoft Azure and ASP.NET. Follow him on Twitter at twitter.com/ReitanErik.

TOM DYKSTRA is a senior content developer at Microsoft, focusing on Microsoft Azure and ASP.NET.

THANKS to the following technical experts for reviewing this article:
Michael Collier, Brady Gaster, John de Havilland, Ryan Jones,
Vijay Ramakrishnan and Pranav Rastogi

msdnmagazine.com



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 and C++. SDKs for multiple platforms. (See site for articles on faceted search, SQL, MS Azure, etc.)

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

Fault Tolerance Pitfalls and Resolutions in the Cloud

Mario Szpuszta and Srikumar Vaitinadin

There are several mechanisms built into Microsoft Azure to ensure services and applications remain available in the event of a failure. Such failures can include hardware failures, such as hard-disk crashes, or temporary availability issues of dependent services, such as storage or networking services. Azure and its software-controlled infrastructure are written in a way to anticipate and manage such failures.

In the event of a failure, the Azure infrastructure (the Fabric Controller) reacts immediately to restore services and infrastructure. For example, if a virtual machine (VM) fails due to a hardware failure on the physical host, the Fabric Controller moves that VM to another physical node based on the same hard disk stored in Azure storage. Azure is similarly capable of coordinating upgrades and updates in such a way as to avoid service downtime.

For computing resources (such as cloud services, traditional IaaS VMs, VM scale sets), the most important and fundamental concepts for enabling high availability are fault domains and upgrade domains. These have been part of Azure since its inception. This article will provide not so well-known clarification around those concepts.

The Azure Datacenter Architecture

To fully understand fault domains and upgrade domains, it helps to visualize a high-level view of how Azure datacenters are structured. Azure datacenters use an architecture referred to within Microsoft as Quantum 10. It supports higher throughput compared to previous datacenter architectures. Its topology implements a

full, non-blocking, meshed network that provides an aggregate backplane with a high bandwidth for each Azure datacenter, as shown in **Figure 1**.

The nodes are arranged into racks. A group of racks then forms a cluster. Each datacenter has numerous clusters of different types. Some clusters are responsible for Storage while others are responsible for Compute, SQL and so on. The Top-Of-Rack (TOR) switch is a single point of failure for the entire rack.

The cluster's Fabric Controller manages all the machines or nodes. The Fabric Controller orchestrates deployments across nodes within a cluster. Every cluster has more than one Fabric Controller for fault tolerance. The Fabric Controller must be aware of the health of every node within the cluster. This helps it determine if the node can run deployments. It also helps the Fabric Controller detect failures so it can automatically heal deployments by re-provisioning the affected VMs on different physical nodes.

To better assist the Fabric Controller in determining the health of a node, every machine running within a cluster has different agents to continuously monitor node health and communicate the same back to the Fabric Controller. It's essential to understand how the different components work together to make this happen. The important components, as shown in **Figure 2**, are:

- **Host OS:** OS running on the physical machine.
- **Host Agent:** A process running on individual nodes that provides a point of communication from that machine to the Fabric Controller.
- **Guest OS:** OS running within the VM.
- **Guest Agent:** Resides in the VM and communicates with Host Agent to monitor and maintain the health of the VM.

Fault Domains and Upgrade Domains

For maintaining high availability of any Platform-as-a-Service (PaaS) application, every PaaS application the Fabric Controller hosts would be spread across different fault domains and update domains.

A fault domain is a physical unit of failure. It's closely related to the physical infrastructure in the datacenters. For Azure, every

This article discusses:

- Enhancing fault tolerance with Microsoft Azure
- How to assign virtual resources for best results
- Maintaining fault tolerance during upgrades

Technologies discussed:

Microsoft Azure, Platform as a Service, Infrastructure as a Service

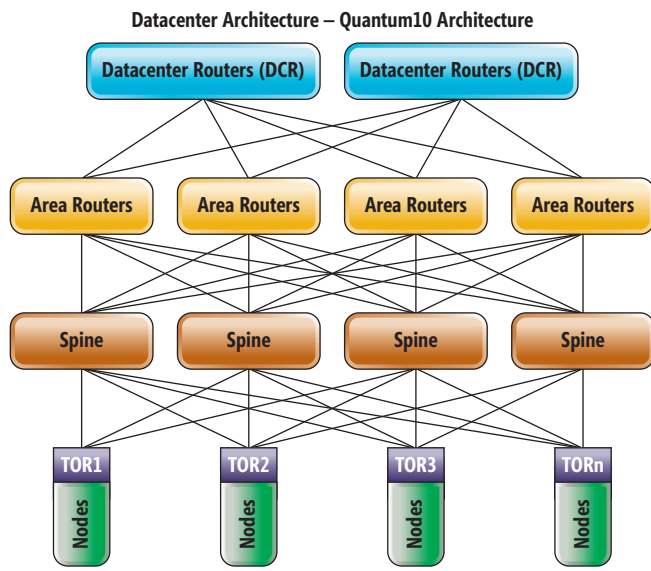


Figure 1 High-Level Azure Datacenter Architecture

rack of servers corresponds to a fault domain. While Azure guarantees any PaaS application (with more than one instance) hosted by the platform would be available across multiple fault domains, the total number of fault domains over which the instances of the application are spread across is determined by the Fabric Controller based on availability of machines within the datacenter.

An upgrade domain is a logical unit that helps maintain application availability when you push updates to the system. For PaaS applications, this is a user-configurable setting. An application on Azure can have its instances spread across a maximum of five upgrade domains (see Figure 3).

Fault Domains, Upgrade Domains and IaaS VMs

To spread Infrastructure-as-a-Service (IaaS) VMs across fault domains and upgrade domains, Azure introduced the concept of Availability Sets. All instances within an availability set are spread across two or more fault domains and assigned separate upgrade domain values.

If you don't assign VMs to an availability set, you're not eligible for service-level agreements (SLAs) for those VMs. It's important to understand this because it defines how you can achieve high availability for your services and applications even if failures happen or upgrades are pushed out to Azure datacenters. Only by assigning your VMs to an availability set, you can avoid being affected by such failures.

To demonstrate the importance of this, consider this scenario: The Azure product team pushes OS updates across all datacenters on a regular basis. To push updates to the entire datacenter, the host OS (physical machines) and the guest OS (VMs hosting PaaS applications or your own IaaS VMs) must be updated to the latest OS. To roll out the updates without affecting availability of the applications:

1. The host OS updates are performed across the datacenter one fault domain at a time, for all available fault domains.
2. Guest OS updates are performed on every user application one upgrade domain at a time, for all the available upgrade domains.

With these approaches, Azure can push upgrades to its own infrastructure while maintaining service availability—as long as

you run at least two instances per service or at least two VMs as part of an availability set (such as a load-balanced Web service, SQL Server AlwaysOn Availability Group Nodes and so on).

How Many Fault Domains?

Fault and upgrade domains help maintain availability when it comes to PaaS-like workloads, which are mostly stateless. Stateless Web applications are compatible with these approaches. Even if a subset of the nodes becomes unavailable during upgrade cycles or temporary downtimes, the overall Web application or service remains available.

The situation gets trickier when it comes to infrastructure of a more stateful nature, such as database servers (be it RDBMS or NoSQL). In these cases, knowing your servers are spread across multiple fault domains might not be enough. A database cluster might require a minimum number of nodes to be up at all times for cluster health. Consider Quorum-based approaches for electing new master nodes in case of failures.

For IaaS VMs, Azure guarantees VMs within the same availability set will be deployed on at least two fault domains (therefore two racks). While there's some probability VMs within an availability set are deployed across more than two fault domains, there's no guarantee. In practical testing over the past few years, the project always used exactly two fault domains when deploying to North or West Europe, as well as several U.S. regions (as shown in Figure 4).

Figure 4 shows the result of a Get-AzureVM command issued through Azure PowerShell, which then displays the result in its GridView control. It shows the VMs within the cluster—which are all part of the same availability set (not shown in the grid)—are deployed across two fault domains and three upgrade domains.

The sql1 and sqlwitness nodes of that sample deployment reside on the same physical rack. The same TOR connects that rack to the rest of the datacenter. The sql2 node sits on a different rack.

Only Two Fault Domains?

For stateless applications such as Web APIs or Web applications, being deployed across only two fault domains shouldn't be a

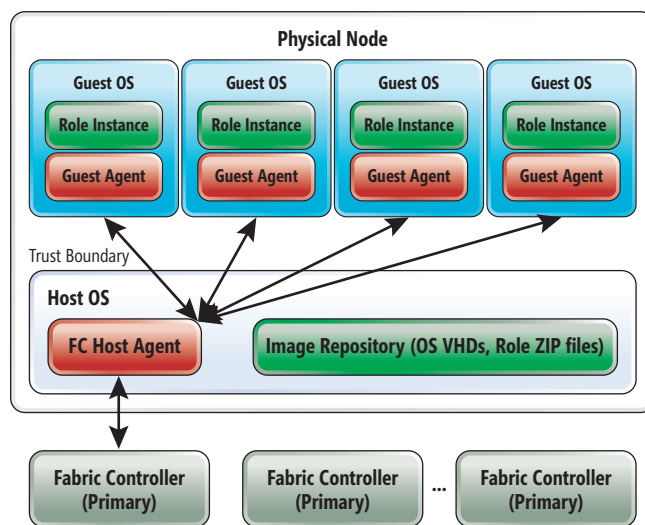


Figure 2 Inside the Physical Machine of a Microsoft Azure Cluster

problem, at least from a consistency perspective. For stateful workloads such as database servers, it's a different story, at least from an availability perspective.

Depending on how a cluster works, it could be important to know how many nodes can go down in case of a failure before it affects the cluster's health. If a cluster depends on quorum votes or majority-based votes for certain operations, such as electing new masters or confirming consistency for read requests, the question of how many nodes can go down in a worst-case scenario is more important.

Even though the fault domain automatically recovers your VMs, the question of how many nodes could fail at the same time is relevant. A recovery operation might take time depending on how long it takes the Fabric Controller to recover the VM itself and the database system running on the VM.

The whole topic becomes more important when you understand the internal behavior of Azure automatism in upgrades. In the case of IaaS VMs, all upgrades to the host OS running on the hypervisor on each of the physical nodes that host your VMs happens based on fault domains—not upgrade domains as assumed in the broad developer community. Upgrade domains are only used for updating applications running inside PaaS VMs. That means you'll be affected by host OS upgrades every quarter, which is the typical interval. If your cluster is deployed across two fault domains, but depends on majority votes and the like, you could have intermittent downtime.

A common example of this is deploying a MongoDB replica set in Azure. Each MongoDB replica set requires exactly one master. If that master node fails, a new master is elected through the remaining nodes. That election requires a majority of votes for electing the master. If not enough nodes are up to vote a new master, the whole replica set is declared an unhealthy state and can be considered as “down.”

The MongoDB documentation (bit.ly/1SxKrYI) clearly states a fault-tolerance per replica set size. Only one node can fail in a replica set of three nodes. In a set of five nodes, two is the maximum number of nodes that can fail without the whole cluster going down, as shown in **Figure 5**.

If the number of database nodes you need to run in a replica set

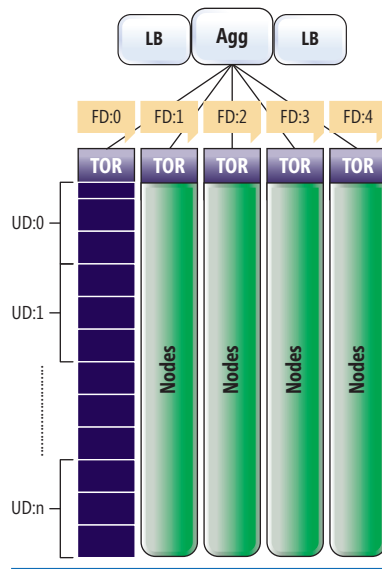


Figure 3 Fault Domain and Upgrade Domain Configuration

is even (for example, two database nodes for a replica set), MongoDB introduced the concept of an arbiter. An arbiter acts as a voting server for elections, but doesn't run the entire database stack (for saving costs and resources). So if you end up with a MongoDB replica set where two database nodes are sufficient, you need a third node—the arbiter—which is only there to provide an additional vote for majority-based master elections in case of failures.

It's a similar situation with the SQL Server AlwaysOn Availability Group, where a majority of nodes is required to elect a new primary node. The principle of a voting-only member is similar. It's just called a witness in the world of SQL Server (instead of arbiter, as they're called in MongoDB).

Considering SQL Server and the deployment shown back in **Figure 4**, it clearly states sql1 and sqlwitness are on one fault domain and sql2 is on another. If fault domain “0” fails,

the master and witness are both down—only sql2 is left. However, sql2 alone is an insufficient majority for electing a new master in the cluster. That means if fault domain “0” fails, your whole cluster is unhealthy.

The situation would be worse if sql1 and sql2 ended up on the same fault domain. Then both database nodes would be down until the Fabric Controller recovered the nodes from a potential failure or completed the host OS upgrade process.

The situation is similar with a MongoDB replica set. The table shown in **Figure 5** from the official MongoDB docs clearly states that in a replica set of three nodes, only one node can fail for the whole cluster to remain active and available. Therefore, it's critical how your nodes are spread across a given number of fault domains. It can affect you in both Azure host OS upgrades and potential failures.

High Availability in Stateful Services

A valid question, then, is how you can achieve high availability when Azure mostly deploys VMs across two fault domains. There are mid-term and short-term answers to this question.

Mid-term the Azure product group is working to improve the situation dramatically. When you deploy version 2 IaaS VMs (based on the new Azure Resource Manager API), Azure can deploy your workloads across a minimum of three fault domains. That's a good reason to use a version 2 VM and the Azure Resource Manager.

Short-term or as long as you're still dependent on traditional Azure Service Management and version 1 IaaS VMs, it's not that simple. Depending on your SLA, recovery time objective (RTO) and recovery point objective (RPO) targets you have two options that

DeploymentName	Name	Label	VM	InstanceStatus	IpAddress	InstanceStateDetails	PowerState	InstanceErrorCode	InstanceFaultDomain	InstanceName	InstanceUpgradeDomain	InstanceSize
sql1	sql1		Microsoft/Windows...	ReadyRole	100.75.88.22		Started		0	sql1	0	Standard_D1
sql1	sql2		Microsoft/Windows...	ReadyRole	100.74.146.106		Started		1	sql2	1	Small
sql1	sqlwitness		Microsoft/Windows...	ReadyRole	100.75.64.58		Started		0	sqlwitness	2	ExtraSmall

Figure 4 Fault Domains for a Sample SQL AlwaysOn AG Cluster

DocuViewware

NEW Universal HTML5 Viewer & Document Management Kit



zero-footprint solution



super-easy integration



fast & crystal clear rendering



fully customizable UI
look & feel



mobile devices optimization



annotations, thumbnails
bookmarks & text search

supports nearly 100 formats

reduce the risk of downtimes. Both approaches are shown in **Figure 6**, and are based on a SQL Server AlwaysOn Availability Group.

The target is always to reduce the impact of both regularly occurring Host OS upgrades and the eventual failure. For a three-node cluster in a single datacenter, deploy one node outside the VM availability set and two nodes as part of the same VM availability set.

The effects of host OS upgrades are nearly completely mitigated with that approach. The timing of upgrades of VMs inside availability sets is different from single VM host OS upgrades. Host OSes running single VMs without availability sets are typically upgraded approximately a week earlier than those with VMs in availability sets.

In the case of fault domain failures, you can only reduce the probability of being affected. There's always a probability the node outside the availability set lands on one of the fault domains of the VMs within the availability set. Much depends on the resources consumed and available in an Azure datacenter.

For an Active Directory Domain as in **Figure 6**, there's no need for such a solution. There's only a primary and backup domain controller for high availability, anyway. That's two nodes perfectly spread across two fault domains, which is what Azure guarantees for version 1 IaaS VMs.

That still leaves you with the challenge of being better prepared for the SQL nodes based on **Figure 6**. You can reduce that risk by not deploying sqlwitness in the same datacenter. That doesn't just reduce the probability; it essentially eliminates the risk. That solution is also expressed in **Figure 6**: Distribute your deployment across two regions.

Two Options

Depending on your SLA, RPO and RTO needs and what you're willing to pay for high availability, again you have two major options: The fully functional backup deployment in a second region, or having just the arbiter/witness in the secondary region.

A fully functional secondary deployment means you replicate your entire deployment in a secondary region. That would also include your front-end and middle-tier applications and services.

Figure 5 MongoDB Replica Set Fault Tolerance

Number of Members	Majority Required to Elect a New Primary	Fault Tolerance
3	2	1
4	3	1
5	3	2
6	4	2

In the event of a major failure in the primary region, you could then redirect customers to the secondary region.

Such deployments are typically built for strong RPO/RTO targets. For database systems such as MongoDB or SQL AlwaysOn with short RPOs and RTOs, such needs typically result in spanning the replica set or SQL AG cluster across

two regions with ongoing replication enabled across those regions. Although the replication across regions will probably be asynchronous due to latency and performance issues, replication will happen in anywhere from milliseconds to minutes, as opposed to double-digit minutes or hours.

On the other hand, running just a witness or arbiter in the secondary region as a single VM is a much cheaper alternative. It's good enough when you just need to keep your primary cluster alive in case of fault domain failure. It doesn't give you the option of immediately failing over to an entire secondary region without some serious additional steps, such as spinning up new nodes and VMs in the secondary region.

In the example shown in **Figure 6**, you could also run a full SQL node in the secondary region as the only node. Because it runs as a single VM, it would have different upgrade cycles. Also, because it runs in a different datacenter, the probability of it being upgraded or failing at exactly the same time as the nodes in the primary availability set is low.

Wrapping Up

Achieving high availability and fault tolerance for your applications and services isn't a simple process. It requires understanding and adjusting to fundamental concepts. You need to understand fault domains, upgrade domains and availability sets. You especially need to understand the fault-tolerance requirements of stateful systems you're using in your infrastructure when moving to Azure. Map those fault-tolerance requirements to behaviors of fault domains and upgrade domains in Azure.

For entirely new IaaS deployments, be sure to leverage IaaS VMs v2 as part of the Azure Resource Manager and Resource Group efforts. That way, you'll benefit from the fault tolerance of being deployed on at least three fault domains. For deployments using traditional service management, make sure you understand and embrace the realities outlined in this article. These suggestions can help reduce impacts of fault domain downtimes and maintenance events such as Host OS Upgrades. By embracing and adjusting to the concepts outlined here, you'll be able to achieve high availability without unwanted surprises. ■

MARIO SZPUSZTA is a principal program manager for the DX Corp. Global ISV team. He works with independent software vendors across the world to enable their solutions and services on Microsoft Azure. You can reach him through his blog (blog.mszoool.com), on Twitter (twitter.com/mszoool) or via marioszp@microsof.com.

SRIKUMAR VAITINADIN is a software development engineer for the DX Corp. Global ISV team. Before that he was involved in migrating Microsoft properties to Azure. You can get in touch with Srikumar via srivaityi@microsoft.com.

THANKS to the following technical experts for reviewing this article: Guadalupe Casuso and Jeremiah Talkar

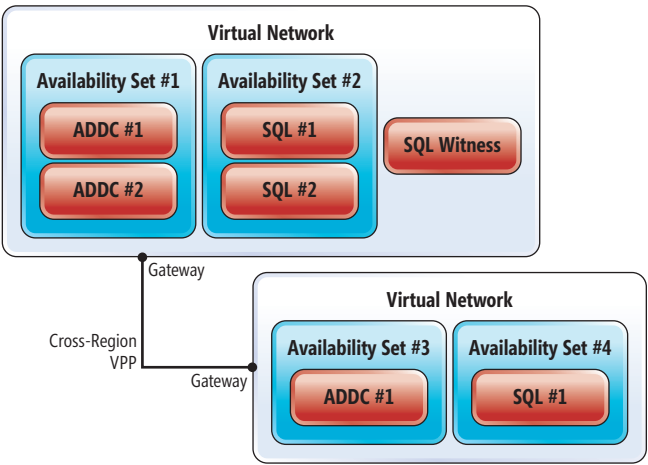


Figure 6 SQL Server AlwaysOn Availability Group Deployment

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!



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.





NAVIGATE THE NET HIGHWAY

DEVELOPMENT TOPICS INCLUDE:

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



*Register by September 2
and Save \$200!*



USE PROMO CODE NYSEP4

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

VSLIVE.COM/NEWYORK

Only 2 Stops Left on the 2015 Ultimate Code Trip!



TURN THE PAGE FOR
MORE EVENT DETAILS.



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 September 2
and Save \$200!**



USE PROMO CODE
VSLSEP4

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

VSLIVE.COM/NEWYORK

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: Embracing DevOps with Visual Studio - Brian Harry, Corporate Vice President, Microsoft			
9:15 AM	10:30 AM	T01 - AngularJS 101 - Deborah Kurata	T02 - Introduction to Next Generation of Azure PaaS – Service Fabric and Containers - 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 - Windows 10 Universal Apps for .NET Developers - Lucian Wischik
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.

Creating Unified, Heroku-Style Workflows Across Cloud Platforms

Bruno Terkaly

A few years ago, Microsoft Azure was just getting off the ground and there was little media coverage. Over the last few years that has changed dramatically. The engineering teams at Microsoft and the community at large have contributed significantly. With this installment of Azure Insider, the series moves to a more customer-centric, case-study view of the world.

For this first Azure case study, I've spoken to Gabriel Monroy. He saw an opportunity and quickly developed it into a technology and a start-up company called Deis. His company was quickly acquired, and Monroy became CTO of the new company. When I first met Monroy and started working with his technology at a hackathon, I told him, "It won't be long before you're acquired." This was back in January 2015. Not more than a few months later, his fledgling company was acquired by EngineYard.

Mother of Invention

There continues to be an explosion of distributed computing platforms that span both on-premises and public clouds. These platforms are powered by container-enabled OSes such as Linux and Windows, Docker containers, and cluster-enabled versions of Linux such as CoreOS.

Most successful open source projects are born out of need. Imagine you're an architect helping the financial community set up large

clusters of virtual machines, trying to support development, testing and product development. Before long, you realize you keep solving the same problem over and over.

That's just what happened to Monroy, who was doing Linux development for the financial community back in 2005 and 2006. He was leveraging some of the earliest technologies around containerization, probably at the same time Solomon Hykes started hacking to create Docker. Much of Monroy's work, in fact, ended up in Docker.

Around that time many companies were struggling with the same need to streamline the development/testing/production pipeline. The ideal was to get to the stage of continuous integration—that elevated state that gets your software to the users in an automated and timely fashion.

Companies wanted a repeatable process, but there were little or no tools. Companies also wanted developer self-service. They didn't want developers held back by the lack of hardware or the tyranny of IT operations. Developers didn't want to pull in ops just to iterate on a new idea or project.

So, instead, developers started to work in a nefarious world of shadow IT—secretly provisioning infrastructure and freeing themselves from the dependency of others. Developers also wanted to be able to operate on any public cloud, whether Amazon Web Services, Digital Ocean, Google or Azure. They also wanted to run on bare metal in their own datacenters, if necessary.

Opportunity Knocks

Back in late-2007 and early-2008, Heroku offered a new approach to distributed computing, focusing on Ruby developers who wanted a single environment to develop, test and deploy applications. Developers wanted to focus on their applications, not on the underlying infrastructures. They wanted a single command-line interface with the underlying platform that would let them focus on just the app and its data. They didn't want to worry about availability, downtime,

This article discusses:

- Streamline the development/testing/production process
- Incorporating Heroku-style workflows on multiple cloud platforms
- Applying container technology to DevOps

Technologies discussed:

Microsoft Azure, Heroku, Node.js, Linux, Docker

Converging Technologies

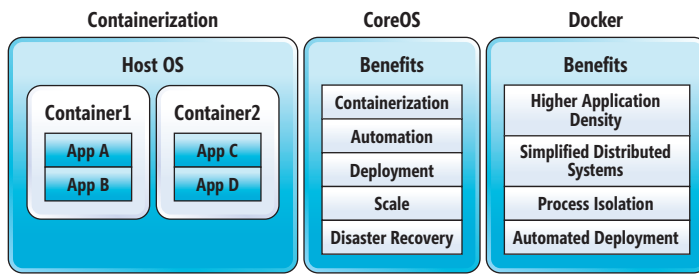


Figure 1 Converging Technologies that Support Deis

disaster recovery, deployment, production, scaling up and down as needed, version control, and all those typical issues. At the same time, they did not want to depend on outside IT administrators to support their workload. That's when Monroy first saw opportunity.

A number of related technologies were coming together that triggered an entrepreneurial spark in Monroy's mind. He could enable Heroku-style developer workflows on multiple cloud platforms. Before diving into all the technologies that enabled Monroy's idea, here's a look at this idyllic world where developers can enjoy Heroku-style workflows on virtually any public cloud with Deis.

The following code installs the Deis platform. This assumes there's a cluster of CoreOS Linux machines with which to work (hosted on-premises or in the cloud):

```
# Install Deis tooling
$ deisctl install platform
# Deis platform is running on a cluster
$ deisctl start platform
$ deis register http://deis.example.com
```

With the exception of a few commands relating to logins and SSH certificates, the development team is ready to leverage Deis and start deploying applications. Once Deis is installed, developers can deploy applications to development, then test and move them to production with just a handful of commands.

Enabling Technologies

Other technologies were coming to fruition at the same time, which helped Deis flourish, as shown in **Figure 1**.

Containerization is a critical technology present in modern server-side OSes. It has been in Linux for some time. While it's not currently present in Windows Server, it should be soon. The concept of containerization is you take a host OS and partition it in multiple dimensions—memory, CPU and disk. You can break one physical computer running one physical OS into multiple containers. Each container is segregated so applications run isolated from each other while sharing the basic host OS.

This increases efficient hardware utilization, because containers can run in parallel without affecting one another. Linux Containers (LXC) isolate CPU, memory, file I/O and network resources. LXC includes namespaces, which isolate the application from the OS and separate the process trees, network access, user IDs and file systems.

Monroy had been leveraging LXC in the early days, even before it was a fundamental part of Docker. Then Docker came along and democratized containerization by standardizing it across Linux distributions. The real breakthrough came when Solomon created a central repository of Docker images. This made available an

ecosystem of publicly available containers other developers could reuse at will. There are more than 14,000 available images available at registry.hub.docker.com.

You can find almost every conceivable application pattern to accelerate your next project. You can even make your own images available through this registry. So if you want to use Nginx or Kafka in your application, you don't need to worry about downloading and installing applications, configuring system settings and generally having to know the idiosyncrasies of individual software applications. Deis curates your applications as Docker images, and then distributes across your cluster as Docker containers. It's easy to compose your

own applications within a container by leveraging a Docker file:

```
FROM centos:latest
COPY . /app
WORKDIR /app
CMD python -m SimpleHTTPServer 5000
EXPOSE 5000
```

Once you've defined your Docker files and provisioned Deis on your cluster, your application deployment and management becomes more simple and powerful. When you combine this with the Git source code repository, it's the best of both worlds. You can use versioning for both application source code, as well as the infrastructure (Docker container) itself.

This style of application development and deployment is repeatable and predictable. It greatly accelerates the ability to move between development, testing and production. A Docker file deploys a simple agent to the server to dev, test or production:

```
# Assume the current folder contains Docker files
$ git add .
$ git commit -m "notes by a developer"
$ git push deis master
```

Back to Heroku

Monroy noticed Heroku was making huge inroads in the developer community because it greatly simplifies application deployment, execution and management, mostly Ruby and Node.js apps.

Developers typically sit on a hosted command prompt that lets them perform virtually all aspects of application development, infrastructure provisioning and scaling tasks. Monroy found that brilliant—a single place for a developer to get all his work done, and minimize the plethora of development tools.

A lot of the management headaches of running a cluster were automated—backup and restore, configuring DNS, defining a load balancer, monitoring disk usage, managing users, platform logging and monitoring, and so on. Adding nodes is a simple case of modifying a URL in a cloud-config file through a command-line interface. Perhaps the most important aspect of running a cluster is to make it self-healing, such that failover and disaster recovery are automatically included.

CoreOS

While Heroku had the funding and the time to build this custom platform, Monroy needed an off-the-shelf solution for managing clusters that included pieces such as load balancing, monitoring and lighting, failover, and so on. At about the same time, one of the Linux distributions known as CoreOS was also gaining developer mindshare.

CoreOS brought the perfect mix of technologies to help facilitate the world Monroy envisioned. CoreOS is an open source Linux OS

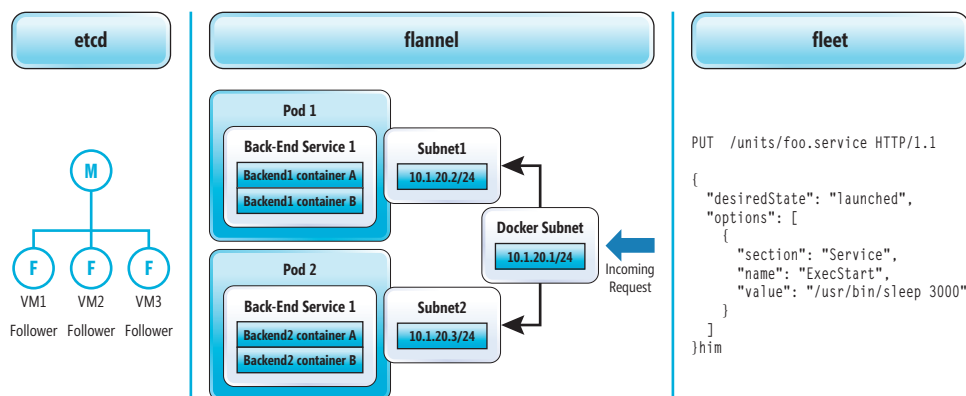


Figure 2 Conceptual Diagram of CoreOS

designed for cluster deployments. It focuses on automation, deployment, security, reliability and scalability. These were precisely the characteristics with which Heroku attracted developers.

CoreOS did indeed provide the silver bullet for which Monroy was looking. CoreOS isn't an ordinary Linux-based OS. Interestingly, it's trying to pioneer its own version of a Docker container called the Rocket container runtime.

The innovation CoreOS brings to the table is significant. Monroy and his team were most interested in etcd, fleet and flannel. Etcd is a distributed key value store that provides a reliable way to store data across a cluster of machines by gracefully handling master elections during network partitions that will tolerate machine failure, including the master. The etcd key value store that stores cluster configuration information is also intelligently distributed throughout the cluster.

An easy-to-use API lets you change values in this configuration file, which is then automatically replicated to other nodes in the cluster. Flannel provides a virtual network that gives a subnet to each pod to be used with each container runtime. This provides each container a unique, routable IP within the cluster. This dramatically reduces port-mapping complexity.

Fleet helps you think of your cluster as a single init system, freeing you from having to worry about the individual machines where your containers are running. Fleet automatically guarantees your container will be running somewhere on the cluster. So if the machine fails or needs updating, the fleet software will automatically move your workload to a qualified machine in the cluster.

Notice in **Figure 2** you can send a put request to the cluster to tell

fleet the desired state of a particular service. That's underlying plumbing that makes up fleet services. This put request on your behalf frees you from having to worry about the details of the cluster.

Monroy now had everything he needed to build Deis—a standardized containerization model from Docker and a cluster-aware version/implementation of Linux called CoreOS. He could now offer Heroku-style development for the masses, not just those who could afford the hefty fees required by

Salesforce.com, the company that now offers Heroku as a service.

Three basic components are a control plane, a Docker register and a data plan. These all work in concert. It starts with the developer using the Git push in a new release, which might include both source code for the application and a Docker build file.

This new build, along with the existing configuration, results in a new release. This is then pushed up to the Docker registry. The scheduler running in the data plan then pulls the released images into dev, test or production.

At this stage, the containers are managed both by CoreOS and by Deis, providing fault-tolerance, scalability, and the other Platform-as-a-Service features. Also in the data plan is the router, which takes denial submitted by the app use and "routes" users to the appropriate container to satisfy the request. **Figure 3** depicts these technologies working together.

Wrapping Up

Some of the most successful open source projects out there don't reinvent the wheel. They take preexisting components, bring them together under a single umbrella and apply technology in a unique way. Monroy and the team at Deis did just that. They harnessed the power of Docker containers, CoreOS and Heroku-style workflows. The team could implement Deis not only on Azure, but also in other public clouds, not to mention on-premises, as well.

In the next installment of Azure Insider case studies, I'll examine Docker. What is Docker? How did it become a billion dollar company in just a few years and how is it changing the way applications are developed, tested and rolled into production? ■

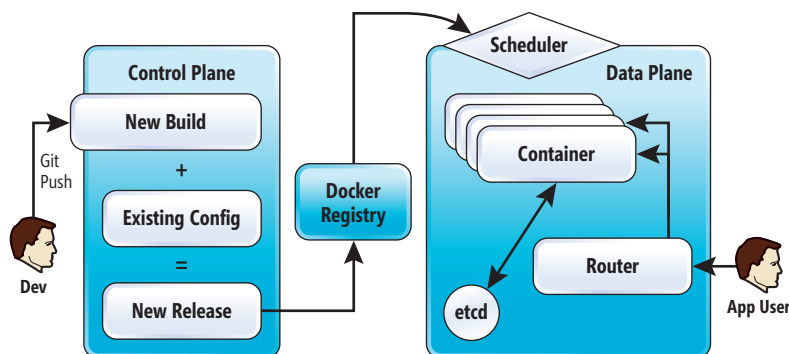


Figure 3 Deis Architecture

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.

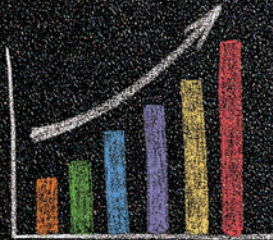
THANKS to the following technical expert for reviewing this article: Gabriel Monroy (EngineYard)

Spreadsheets Made Easy.



Fastest Calculations

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



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.



Windows
Forms



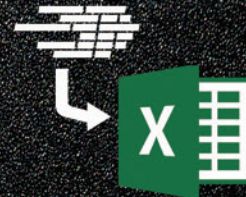
Silverlight



WPF

Powerful Controls

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



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

Streamline Code with Native Profile-Guided Optimization

Hadi Brais

Often a compiler can make bad optimization decisions that don't actually improve the code's performance or, even worse, degrade it when it runs. The optimizations discussed in the first two articles are essential for your application's performance.

This article covers an important technique called profile-guided optimization (PGO) that can help the compiler back end optimize code more efficiently. Experimental results show performance improvements from 5 percent to 35 percent. Also, when used carefully, this technique will never degrade your code's performance.

This article builds on the first two parts (msdn.microsoft.com/magazine/dn904673 and msdn.microsoft.com/magazine/dn973015). If you're new to the concept of PGO, I recommend you first read the Visual C++ Team Blog post at bit.ly/1fJn1DI.

Introducing PGO

One of the most important optimizations a compiler performs is function inlining. By default, the Visual C++ compiler inlines a

function as long as the caller size doesn't grow too large. Many function calls are expanded, however, this is only useful when the call happens frequently. Otherwise, it just increases the size of the code, wasting space from instruction and unified caches and increases the app working set. How would the compiler know whether the call happens frequently? It eventually depends on the arguments passed to the function.

Most optimizations lack reliable heuristics needed to make good decisions. I've seen many cases of bad register allocation that results in significant performance degradation. When compiling the code, all you can do is hope all performance improvements and degradations from all optimizations add up to a positive speed result. This is almost always the case, but it can generate an excessively large executable.

It would be nice to eliminate such degradations. If you could tell the compiler how the code is going to behave at run time, it could better optimize the code. The process of recording information on program behavior at run time is called profiling and the generated information is called a profile. You can provide one or more profiles to the compiler, which will use them to guide its optimizations. This is what the PGO technique entails.

You can use this technique on both native and managed code. However, the tools are different and so here I'll discuss only native PGO and leave managed PGO for another article. The rest of this section discusses how to apply PGO to an app.

PGO is a great technique. Like everything else, though, it has a downside. It takes some time (depending on the app size) and effort. Fortunately, as you'll see later, Microsoft provides tools that can substantially reduce the time to perform PGO to an app. There

This article discusses:

- Generating PGO database files to guide optimization
- Building instrumentation into your code
- Examining the code to spot optimized differences

Technologies discussed:

Visual Studio 2013, Visual C++

Code download available at:

bit.ly/1gpEaCY

are three phases to perform PGO to an app—the instrumentation build, training and the PGO build.

The Instrumentation Build

There are several ways to profile a running program. The Visual C++ compiler uses static binary instrumentation, which generates the most accurate profiles, but takes more time. Using instrumentation, the compiler inserts a small number of machine instructions at locations of interest in all functions of your code, as shown in **Figure 1**. These instructions record when the associated part of your code has been executed and include this information in the generated profile.

There are several steps to building an instrumented version of an app. First, you have to compile all source code files with the `/GL` switch to enable Whole Program Optimization (WPO). WPO is required to instrument the program (it's not technically necessary, but it helps make the generated profile much more useful). Only those files that have been compiled with `/GL` will be instrumented.

For the next phase to go as smooth as possible, avoid using any compiler switches that result in extra code. For example, disable function inlining (`/Ob0`). Also, disable security checks (`/GS-`) and remove runtime checks (`/RTC-`). This means you shouldn't use the default Release and Debug modes of Visual Studio. For files not compiled with `/GL`, optimize them for speed (`/O2`). For instrumented code, specify at least `/Og`.

Next, link the generated object files and the required static libraries with the `/LTCG:PGI` switch. This makes the linker perform

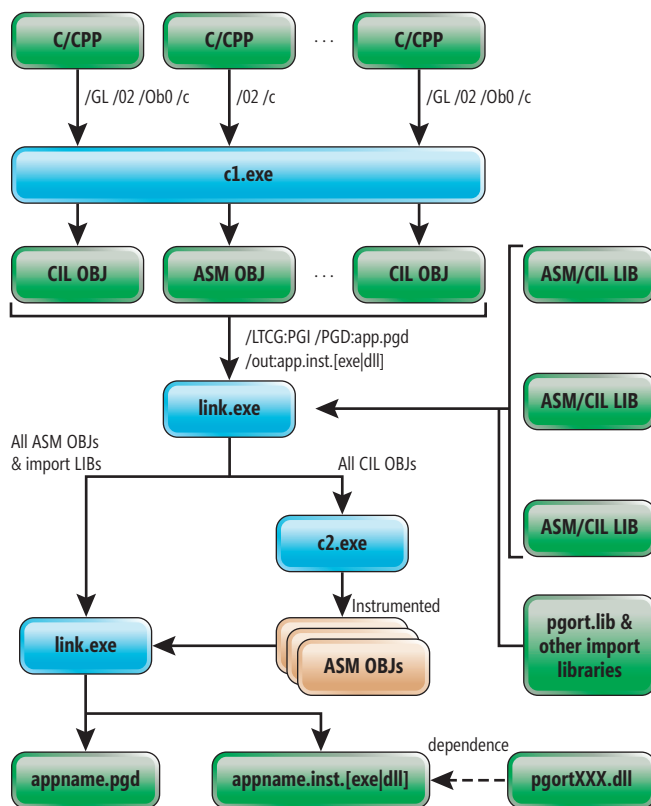


Figure 1 The Instrumentation Build of a Profile-Guided Optimization App

three tasks. It tells the compiler back end to instrument the code and generate a PGO database (PGD) file. This will be used in the third phase to store all profiles. At this point, the PGD file doesn't contain any profiles. It only has information to identify the object files used to detect at the time of using the PGD file whether they have changed. By default, the PGD file name takes the name of the executable. You can also specify a PGD file name using the optional `/PGD` linker switch. The third task is to link the `pgort.lib` import library. The output executable is dependent on the PGO runtime DLL `pgortXXX.dll` where `XXX` is the version of Visual Studio.

The end result of this phase is an executable (EXE or DLL) file bloated with instrumentation code and an empty PGD file to be filled and used in the third phase. You can only have a static library instrumented if that library is linked to a project to be instrumented. Also, the same version of the compiler must produce all CIL OBJ files; otherwise, the linker will emit an error.

Profiling Probes

Before moving on to the next phase, I'd like to discuss the code the compiler inserts to profile the code. This lets you estimate the overhead added to your program and understand the information being collected at run time.

To record a profile, the compiler inserts a number of probes in every function compiled with `/GL`. A probe is a small sequence of instructions (two to four instructions) consisting of a number of push instructions and one call instruction to a probe handler at the end. When necessary, a probe is wrapped by two function calls to save and restore all XMM registers. There are three types of probes:

- **Count probes:** This is the most common type of probe. It counts the number of times a block of code executes by incrementing a counter every time it's executed. It's also the cheapest in terms of size and speed. Each counter is 8 bytes in size on x64 and 4 bytes on x86.
- **Entry probes:** The compiler inserts an entry probe at the beginning of every function. The purpose of this probe is to tell the other probes in the same function to use the counters associated with that function. This is needed because probe handlers are shared between probes across functions. The entry probe of the main function initializes the PGO runtime. An entry probe is also a count probe. This is the slowest probe.
- **Value probes:** These are inserted before every virtual function call and switch statement and used to record a histogram of values. A value probe is also a count probe because it counts the number of times each value appears. The size of this probe is the largest.

A function won't be instrumented by any probe if it only has one basic block (a sequence of instructions with one entry and exit). In fact, it might be inlined in spite of the `/Ob0` switch. Besides the value probe, every switch statement causes the compiler to create a constant COMDAT section that describes it. The size of this section is approximately the number of cases multiplied by the size of the variable controlling the switch.

Every probe ends with a call to the probe handler. The entry probe of the main function creates a vector of (8 bytes on x64 and

4 bytes on x86) probe handler pointers where each entry points to a distinct probe handler. In most cases, there will be only a few probe handlers. Probes are inserted in each function in the following locations:

- An entry probe at the entry of the function
- A count probe in every basic block ending with a call or ret instruction
- A value probe just before every switch statement
- A value probe just before every virtual function call

So the amount of memory overhead of the instrumented program is determined by the number of probes, the number of cases in all switch statements, the number of switch statements and the number of virtual function calls.

All probe handlers at some point advance a counter by one to record execution of the corresponding block of code. The compiler uses the ADD instruction to increment a 4-byte counter by one and on x64, the ADC instruction to add the carry to the high 4 byte of the counter. These instructions aren't thread-safe. This means all probes by default aren't thread-safe. If at least one of the functions might be executed by more than one thread at the same time, the results won't be reliable. In this case, you can use the /pogosafemode linker switch. This causes the compiler to prefix these instructions with LOCK, which makes all probes thread-safe. Of course, this makes them slower, as well. Unfortunately, you can't selectively apply this feature.

If your application consists of more than one project whose output is either an EXE or a DLL file for PGO, you must repeat the process for each.

The Training Phase

After that first phase, you have an instrumented version of the executable and a PGD file. The second phase is training, in which the executable will generate one or more profiles to store in a separate PGO Count (PGC) file. You'll use these files in the third phase to optimize the code.

This is the most important phase because profile accuracy is crucial to the success of the whole process. For a profile to be useful,

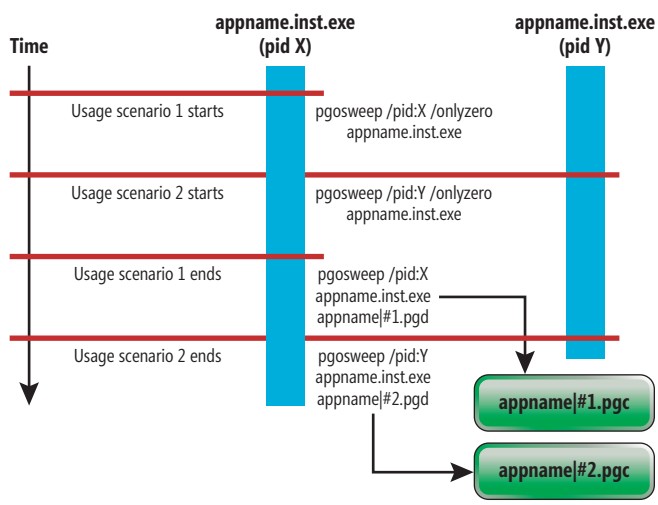


Figure 2 The Training Phase of Creating a PGO App

it has to reflect a common usage scenario of the program. The compiler will optimize the program assuming the exercised scenarios are common. If this wasn't the case, your program might perform worse in the field. A profile generated from a common-usage scenario helps the compiler know the hot paths to optimize for speed and the cold paths to optimize for size, as shown in **Figure 2**.

The complexity of this phase depends on the number of usage scenarios and the nature of the program. Training is easy if the program doesn't require any user input. If there are many usage scenarios, sequentially generating a profile for each scenario might not be the quickest way.

The compiler will optimize the program assuming the exercised scenarios are common.

In the complex training scenario shown in **Figure 2**, pgosweep.exe is a command-line tool that lets you control the contents of the profile maintained by the PGO runtime when it runs. You can spawn more than one instance of the program and concurrently apply usage scenarios.

Imagine you have two instances running in processes X and Y. When one scenario is about to start on process X, call pgosweep and pass to it the process ID and the /onlyzero switch. This causes the PGO runtime to clear the part of the in-memory profile for that process only. Without specifying the process ID, the whole PGC profile will be cleared. Then the scenario can start. You can initiate usage scenario two on process Y in a similar fashion.

The PGC file will be generated only when all running instances of the program terminate. However, if the program has a long startup time and you don't want to run it for every scenario, you can force the runtime to generate a profile and clear the in-memory profile to prepare it for another scenario in the same run. Do this by running pgosweep.exe and passing the process ID, the executable file name and the PGC file name.

By default, the PGC file will be generated in the same directory as the executable. You can change this with the VCPROFILE_PATH environment variable, which has to be set before you run the first instance of the program.

I've discussed the data and instruction overhead of instrumenting code. In most cases, this overhead can be accommodated. The memory consumption of the PGO runtime by default will not exceed a certain threshold. If it turns out that it requires more memory, an error will occur. In this case, you can use the VCPROFILE_ALLOC_SCALE environment variable to raise that threshold.

The PGO Build

Once you've exercised all common usage scenarios, you'll have a set of PGC files you can use to build the optimized version of the program. You can discard any PGC file you don't want to use.

The first step in building the PGO version is to merge all PGC files with a command-line tool called pgomgr.exe. You can also

We know how application development can be.



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

**Atalsoft
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.

use this to edit a PGD file. To merge the two PGC files into the PGD file generated in the first phase, run `pgomgr` and pass the `/merge` switch and the PGD file name. This will merge all PGC files in the current directory whose names match the name of the specified PGD file followed by `!#` and a number. The compiler and linker can use the resulting PGD file to optimize the code.

You can capture a more common or important usage scenario with the `pgomgr` tool. To do this, pass the corresponding PGC file name and the `/merge:n` switch, where `n` is some positive integer indicating the number of copies of the PGC file to include in the PGD file. By default, `n` is one. This multiplicity causes a specific profile to bias the optimizations in its favor.

The second step is to run the linker passing the same set of object files as in phase one. This time, use the `/LTCG:PGO` switch. The linker will look for a PGD file with the same name as the executable in the current directory. It will ensure the CIL OBJ files didn't change since generating the PGD file in phase one, and then pass it to the compiler to use and optimize the code. This process is shown in **Figure 3**. You can use the `/PGD` linker switch to explicitly specify a PGD file. Don't forget to enable function inlining for this phase.

Most of the compiler and linker optimizations will become profile-guided. The end result of this phase is a highly optimized executable in terms of size and speed. It's a good idea now to measure your performance gains.

Maintain the Code Base

If you make any changes to any input files passed to the linker with the `/LTCG:PGI` switch, the linker will refuse to use the PGD file when `/LTCG:PGO` is specified. That's because such changes can significantly affect the usefulness of the PGD file.

One option is to repeat the three phases discussed previously to generate another compatible PGD. However, if the changes were tiny (such as adding a small number of functions, calling a function a little less or a little more frequently, or maybe adding a feature that isn't commonly used) then it isn't practical to repeat the whole process. In this case, you can use the `/LTCG:PGU` switch instead of the `/LTCG:PGO` switch. This tells the linker to skip compatibility checks over the PGD file.

These tiny changes will accumulate over time. You'll eventually reach a point where it's beneficial to instrument the app again. You can determine when you've reached this point by looking at the compiler output when PGO building the code. It will tell you how much of the code base the PGD covers. If profile coverage drops to less than 80 percent (as shown in **Figure 4**), it's a good idea to instrument the code again. However, this percentage is highly dependent on the nature of the application.

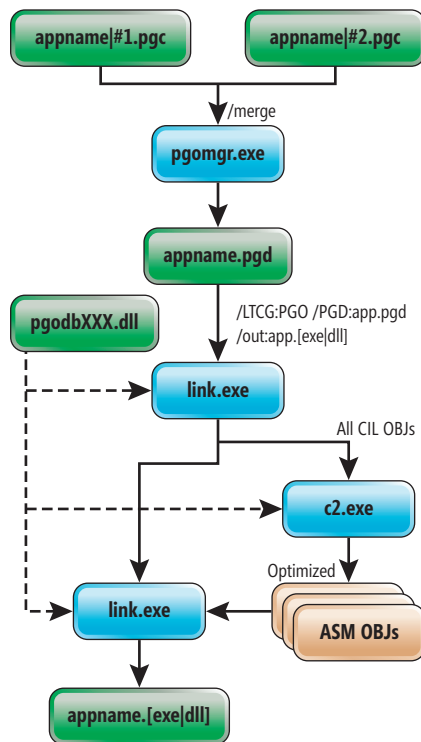


Figure 3 The PGO Build in Phase Three

PGO in Action

PGO guides optimizations employed by the compiler and the linker. I'll use the NBody simulator to demonstrate some of its benefits. You can download this application from bit.ly/1gpEaCY. You'll have to also download and install the DirectX SDK at bit.ly/1LQnKge to compile the application.

First, I'll compile the application in Release mode to compare it to the PGO version. To build the PGO version of the app, you can use the Profile-Guided Optimization menu item of the Build menu of Visual Studio.

You should also enable assembler output using the `/FA[c]` compiler switch (don't use `/FA[c]`s for this demo). For this simple app, it's sufficient to train the instrumented app once to generate one PGC file and use it to optimize the app. By doing this, you'll be having two executables: one blindly optimized and another PGO. Make sure you can access the final PGD file because you'll need it later.

Now if you run both executables one after the other and compare the attained GFLOP counts, you'll notice they achieved similar performance. Apparently, to apply PGO to

the app was a waste of time. Upon further investigation, it turns out that the size of the app has been reduced from 531KB (for the blindly optimized app) to 472KB (for the PGO-based app), or 11 percent. So when you apply PGO to this app, it was reduced in size while maintaining the same performance. How did this happen?

Consider the 200-line function `DXUTParseCommandLine` from the `DXUT/Core/DXUT.CPP` file. By looking at the generated assembly code of the Release build, you can see the size of the binary code is approximately 2700 bytes. On the other hand, the size of the binary code in the PGO build is no more than 1650 bytes. You can see the reason for this difference from the assembly instruction that checks the condition of the following loop:

```
for( int iArg = iArgStart; iArg < nNumArgs; iArg++ ) { ... }
```

The blindly optimized build produced the following code:

```
0x044 jge block1
; Fall-through code executed when iArg < nNumArgs
; Lots of code in between
0x362 block1:
; iArg >= nNumArgs
; Lots of other code
```

On the other hand, the PGO build produced the following code:

```
0x043 j1 block1
; taken 0(0%), not-taken 1(100%)
block2:
; Fall-through code executed when iArg >= nNumArgs
0x05f ret 0
; Scenario dead code below
0x000 block1:
; Lots of other code executed when iArg < nNumArgs
```

Many users prefer to specify parameters in the GUI instead of passing them in the command line. So the common scenario here, as indicated by the profile information, is the loop never iterates. Without a profile, it's impossible for the compiler to know that. So it goes ahead

and aggressively optimizes the code within the loop. It expands many functions resulting in pointless code bloat. In the PGO build, you provided the compiler a profile saying the loop has never executed. From this, the compiler understood there's no point in inlining any functions called from the body of the loop.

You can see another interesting difference from the assembly code snippets. In the blindly optimized executable, the branch that rarely executes is in the fall-through path of the conditional instruction. The branch that's almost always executed is located 800 bytes away from the conditional instruction. This not only causes the processor branch predictor to fail, but also guarantees an instruction cache miss.

The PGO build avoided both of these problems by swapping branch locations of the branches. In fact, it moved the branch that rarely executes to a separate section in the executable, thereby improving working set locality. This optimization is called dead code separation. It would have been impossible to perform without a profile. Infrequently called functions, such as tiny differences in binary code, can make significant performance differences.

When building the PGO code, the compiler will show you how many functions have been compiled for speed of all the instrumented functions. The compiler also shows you this in the Visual Studio Output windows. Typically, no more than 10 percent of the functions would be compiled for speed (think of aggressive inlining), while the rest are compiled for size (think of partial or no inlining).

Consider a slightly more interesting function, DXUTStaticWndProc, which is defined in the same file. The functions control structure looks like this:

```
if (condition1) { /* Lots of code */ }
if (condition2) { /* Lots of code */ }
if (condition3) { /* Lots of code */ }
switch (variable) { /* Many cases with lots of code in each */ }
if-else statement
return
```

The blindly optimized code emits each code block in the same order as in the source code. However, the code in the PGO build has been cleverly rearranged using the execution frequency of each block and the time at which each block was executed. The first two conditions were rarely executed, so to improve cache and memory utilization, the corresponding code blocks are now in a separate section. Also, the functions recognized as falling in the hot path (such as DXUTIsWindowed) are now in-lined:

```
if (condition1) { goto dead-code-section }
if (condition2) { goto dead-code-section }
if (condition3) { /* Lots of code */ }
/* Frequently executed cases pulled outside the switch statement */
if-else statement
return
switch(variable) { /* The rest of cases */ }
```

Additional Resources

For more information on profile-guided optimization databases, check out the blog post from Hadi Brais at bit.ly/1KBcfffQ.

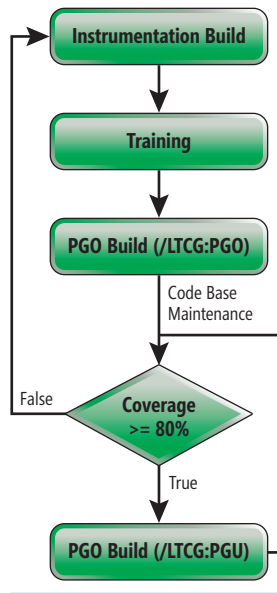


Figure 4 PGO Code Base Maintenance Cycle

Most optimizations benefit from a reliable profile and others become possible to perform. If PGO doesn't result in a noticeable performance improvement, it will certainly reduce the size of the generated executables and their overhead on the memory system.

PGO Databases

The benefits of the PGD profile far exceed guiding the optimizations of the compiler. While you can use the `pgomgr.exe` to merge multiple PGC files, it also serves a different purpose. It offers three switches that let you view the contents of the PGD file to gain full understanding of the behavior of your code with respect to the exercised scenarios. The first switch `/summary` tells the tool to emit a textual summary of the PGD file contents. The second switch `/detail`, used in conjunction with the first, tells the tool to emit a detailed textual profile description. The last switch `/unique` tells the tool to undecorate the function names (particularly useful for C++ code bases).

Programmatic Control

There's one last feature worth mentioning. The `pgobootrun.h` header file declares one function called `PgoAutoSweep`. You can call this function to programmatically generate a PGC file and clear the in-memory profile to prepare for the next PGC file. The function takes one argument of type `char*` referring to the name of the PGC file. To use this function, you have to link the `pgobootrun.lib` static library. Currently, this is the only programmatic support related to PGO.

Wrapping Up

PGO is an optimization technique that helps the compiler and linker make better optimization decisions by referring to a reliable profile whenever a trade-off of size versus speed requires resolution. Visual Studio provides visual access to this technique through the Build menu or the context menu of the project.

However, you can get a richer set of features using the PGO plugin, which you can download from bit.ly/1Ntg4Be. This is also well documented at bit.ly/1RLjPDi. Recall the coverage threshold from Figure 4, the easiest way to get it is with the plug-in as discussed in the documentation. However, if you prefer using the command-line tools, you can refer to the article at bit.ly/1QYT5n0 for plenty of examples. If you have a native code base, it might be a good idea now to try this technique. When you do that, feel free to let me know how it affected the size and speed of your application. ■

HADI BRAIS is a Ph.D. scholar at the Indian Institute of Technology Delhi (IITD), researching compiler optimizations for the next-generation memory technology. He spends most of his time writing code in C/C++/C# and digging deep into runtimes and compiler frameworks. He blogs at hadibraais.wordpress.com. Reach him at hadi.b@live.com.

THANKS to the following Microsoft technical expert for reviewing this article:
Ankit Asthana

Protect Sensitive Information with Azure Key Vault

Rahul Nath

Many applications built today deal with sensitive information in some form or another, often connection strings to an external system such as a database. These keys are typically deployed as part of the application's configuration file and are available as plain text to anyone who can access the deployment servers. Clearly, this poses a huge security threat.

Microsoft Azure Key Vault is a cloud-hosted, hardware security modules (HSM)-backed service for managing cryptographic keys and other sensitive information—secrets in Key Vault terminology—in a central location, and can be accessed via a REST API by specifying the appropriate URI for the key or secret. Key Vault also allows you to create a software key, instead of an HSM-backed one; in both cases the private portion of the key doesn't leave the Key Vault boundary and can never be seen or shared.

In this article, I'll walk you through a scenario in which a large company, Contoso, is building a line-of-business (LOB) application, with development outsourced to various vendors. The application interacts with multiple third-party services and deals with sensitive information, including customer personally identifiable information (PII). I'll look first at the existing implementation and the security issues this poses for the company, then show how the company uses Azure Key Vault to solve these issues.

This article discusses:

- Common problems in storing sensitive information in applications today
- The benefits of using Azure Key Vault
- Setting up and using Azure Key Vault to secure cloud-based applications

Technologies discussed:

Microsoft Azure, Windows PowerShell

The Existing Application

Contoso is a large car manufacturing company currently building an LOB application to provide a connected car experience for customers, vehicles and other interested parties on a single platform. Because Contoso has only a small IT team in-house, it outsourced the development of the application to various vendors, each building different parts of the application. The application is cloud-hosted and exposes different APIs for various entities to interact with it. It also uses third-party application APIs to consume data from external services, and calls into internal Contoso applications exposed via Web services.

The application deals with PII, which is confidential and should be stored securely. Initially, not much thought was given to how sensitive information should be handled, and various vendors had their own approaches. Most of the connection strings were part of the application configuration file so it could be changed per deployment; some keys used for encrypting PII information were hardcoded into the application; and some of the certificate details to connect with the Contoso Web services were in the application's database. This became unmanageable for Contoso, and the company ended up sharing the sensitive information with the application development vendors, which isn't recommended. Contoso also realized that anybody who had access to their deployment servers could very easily tamper with the PII data, which would be a serious threat. The company was forced to unify the approach for maintaining and accessing sensitive information.

The Application Refit

Contoso looked into various alternatives for managing sensitive information and unifying the experience so it could be made consistent across different applications. The company wanted a central store for its keys that could be easily managed by its security team, and it wanted to provide different applications with

different access policies to the information. Moreover, Contoso didn't want any of this to require day-to-day monitoring with a lot of maintenance overhead.

The company chose Azure Key Vault, which allows the use of software and HSM-backed keys, as well as importing existing keys from a PFX file. Key Vault allows you to store small bits of information (secrets) securely in the cloud and access them as required. Because this service is completely managed and maintained on the Azure platform, you are freed from all the overhead of maintaining it.

Key Vault is built in such a way that the keys can't be seen or extracted by anyone, making it a very secure platform. It provides a rich API for accessing and interacting with the vault, as well as a C# API library for applications that run on the Microsoft .NET Framework. The ability to control how different applications and vendors can interact with it is a significant factor that makes Key Vault a good choice for Contoso, as the company wants to restrict the ways various application vendors can interact with the secure information. Moreover, Contoso engineers already heavily use Windows PowerShell scripts for managing their various other servers, and Key Vault is completely manageable through Windows PowerShell.

Creating the Key Vault: The security administrator of the Contoso IT team is responsible for setting up the Key Vault and managing the keys and secrets. Azure Key Vault can be created and managed using Windows PowerShell cmdlets and is available with the latest Azure PowerShell (0.9.2 or higher). Once connected to Azure subscriptions at the Azure PowerShell prompt, you need to switch to AzureResourceManager mode, which the Key Vault cmdlets require. Then, using the New-AzureKeyVault cmdlet, you can create a new vault by specifying a vault name (which is globally unique), resource group and location:

```
Switch-AzureMode AzureResourceManager
New-AzureResourceGroup -Name 'ContosoResourceGroup' -Location 'East Asia'
New-AzureKeyVault -VaultName 'ContosoKeyVault' -ResourceGroupName
'ContosoResourceGroup' -Location 'East Asia'
```

On successful execution of the commands, the details of the newly created key vault are output to the Azure PowerShell console, which includes the vault name and the URI to uniquely identify the vault.

Setting up the Key Vault: Azure Key Vault enables creation and storage of cryptographic keys and also the storage of secrets, which are limited-size octet objects that have no specific semantics.

Cryptographic keys in the vault are represented as JSON Web Key objects and currently only RSA keys are supported. Once a key

is created in the vault, only the public portion of the key is available outside the vault boundary.

A key vault can contain both keys and secrets, and external access to these can be controlled separately. Both keys and secrets are versioned objects in the vault and a new one is uniquely identified by the vault URL and the object name and its version number. Whenever an object is created with an existing name, a new object with the same name and a new version number is created and this becomes the current version. Trying to access an object without a version number returns the current version. The object identifier, which is used to uniquely identify an object within a vault, takes the following format, and is used to interact with keys and secrets using the API:

```
https://(keyvault-name).vault.azure.net/(object-type)/(object-name)/(object-version)
Where:
keyvault-name      : Globally unique key vault name
object-type        : Either "keys" or "secrets," indicating the type of object
object-name        : Unique name within a key vault
object-version      : System generated string, optionally used to
                     identify a specific version of object
```

Keys in Azure Key Vault are represented as JSON Web Key (JWK) objects, with the base JWK specifications extended to enable key types that are unique to the Azure Key Vault implementation. Currently Azure Key Vault supports only the RSA algorithm, an asymmetric algorithm. Azure Key Vault supports Create, Import, Update, Delete, List, Get, Backup and Restore operations on keys, and these operations are available using the REST API and Windows PowerShell cmdlets. Because the Contoso application uses keys to encrypt and decrypt certain PII information, a key needs to be created in the vault. Contoso engineers use the following script to add a new key to the Vault, with a name that's unique within their vault:

```
Add-AzureKeyVaultKey -VaultName 'ContosoKeyVault' -Name 'ContosoPIIKey'
-Destination 'Software'
```

On successful execution this creates a new software RSA key in the ContosoKeyVault, which can be identified with a unique identifier (for example, <https://ContosoKeyVault.vault.azure.net/keys/ContosoPIIKey/bfacf5f768ae42ffb0a0bca448aead87>), which the engineers will share with the application vendors.

Secrets in Azure Key Vault are octet sequences with a maximum size of 25K each; any type of data is accepted and stored securely. The Key Vault supports Create, Get, List, Delete and Update operations on secrets, and these operations are available using the REST API and Windows PowerShell cmdlets. Because the Contoso application saves connection strings in the application configuration files, Contoso engineers decide to move this over to the Key Vault, where it can be stored securely and still be accessible using a unique identifier. The following script was used to add their SQL database connection string to the vault:

```
$ContosoSQLConnectionString = ConvertTo-SecureString -String
"ContosoSQLConnectionString"
-AsPlainText -Force
Set-AzureKeyVaultSecret -VaultName "ContosoKeyVault" -Name
"ContosoSQLConnectionString"
-SecretValue $ContosoSQLConnectionString
```

On successful execution this creates a secret value in the ContosoKeyVault, which can be identified using a unique identifier (such as <https://ContosoKeyVault.vault.azure.net/secrets/ContosoSQLConnectionString/90018dbb96a84117a0d2847ef8e7189d>), which again the engineers share with the application vendors.

Figure 1 Connecting to Key Vault

```
var keyVaultClient = new KeyVaultClient(async (authority, resource, scope) =>
{
    string azureAdApplicationId =
        ConfigurationManager.AppSettings["AzureAdApplicationId"];
    string certificateThumbprint =
        ConfigurationManager.AppSettings["CertificateThumbprint"];
    X509Certificate2 certificate =
        GetCertificateByThumbprint(certificateThumbprint);
    var clientAssertionCertificate =
        new ClientAssertionCertificate(azureAdApplicationId, certificate);
    var authenticationContext = new AuthenticationContext(authority);
    var result =
        await authenticationContext.AcquireTokenAsync(resource,
            clientAssertionCertificate);
    return result.AccessToken;
});
```


Authenticating applications to use Key Vault: The newly created vault is currently accessible only from the Azure account that created it and not by anyone else. Contoso needs to give various client applications access to the vault. Access to an Azure key vault is secured using an Azure Active Directory (Azure AD) application token. To do this, the Contoso engineers first need to create an application in the Azure AD and secure it with either an authentication key (a shared secret) or a certificate. Creating an Azure AD application secured using authentication key access can be accomplished using the Azure Management Portal, under the Applications tab in the Active Directory option. But securing its key vault with an authentication key is not what Contoso prefers, as that would again mean putting sensitive information in the client application configuration file, which is something they wanted to avoid in the first place. Also, because the key vault is the single store for all its sensitive information, allowing access to the credentials to it is something that should be strongly avoided. Therefore, Contoso decides to use certificate-based authentication, so it can deploy the certificate directly to the application requiring it and further can protect the certificate using a password.

Creating an Azure AD application that uses certificate authentication can be done via the Windows PowerShell command line. Contoso already has a mechanism to generate certificates for securing its other internal applications, so the company uses the same service to generate a certificate for the Azure AD application. Once the company has the certificate, the Azure AD application can be created using the following script:

```
$certificateFilePath = "C:\certificates\ContosoADApplication.cer"
$x509Certificate2 = New-Object System.Security.Cryptography.
X509Certificates.X509Certificate2
$x509Certificate2.Import($certificateFilePath)
$rawCertData = $x509Certificate2.GetRawCertData()
$credentialValue = [System.Convert]::ToBase64String($rawCertData)
$startDate = [System.DateTime]::Now
$endDate = $startDate.AddYears(1)
$newADApplication = New-AzureADApplication -DisplayName
"ContosoKeyVaultADApplication"
-HomePage "http://www.contoso.com" -IdentifierUri "http://www.contoso.com"
-KeyValue $credentialValue -KeyType "AsymmetricX509Cert" -KeyUsage "Verify"
-StartDate $startDate -EndDate $endDate
```

With the key vault and the Azure AD application created, Contoso now has to link these together so the application token received from the Azure AD application can be used to authenticate against the key vault. This can be done using the Set-AzureKeyVaultAccessPolicy cmdlet. You also set the permissions the Azure AD application has on the key vault at this time, using the PermissionToKeys and PermissionToSecrets parameters.

The PermissionToKeys parameter specifies an array of key operation permissions to be granted to the application and has a list of acceptable values (Decrypt, Encrypt, UnwrapKey, WrapKey, Verify, Sign, Get, List, Update, Create, Import, Delete, Backup, Restore, All). The PermissionToSecrets parameter specifies an array of secret operation permissions to be granted to the application and also has a list of acceptable values (Get, List, Set, Delete, All). The permissions are applicable to all the keys and secrets in the vault; giving selective permissions to certain keys or secrets in the vault isn't allowed. If selective permission to keys or secrets is required, you need to create separate vaults with the selected keys and secrets to manage the permissions. There is no setup fee for the key

vault itself, but this could change and you would want to check key vault pricing before creating separate vaults. To give different access levels to the same keys and secrets, you can create multiple Azure AD applications and register them with different permission levels.

Contoso engineers use the following script to associate the Azure AD application and the key vault, and to specify the permissions the application has on the vault:

```
$ServicePrincipal = New-AzureADServicePrincipal -ApplicationId
$newADApplication.ApplicationId
Set-AzureKeyVaultAccessPolicy -VaultName 'ContosoKeyVaultRahul'
-ObjectId $ServicePrincipal.Id
-PermissionsToKeys encrypt,decrypt,get -PermissionsToSecrets get
$ServicePrincipal.ApplicationId
```

Connecting to Key Vault from client application: Once the Azure AD application has been created and associated with the key vault, you can use it from a client application to authenticate with the key vault. Because the Contoso application is developed on the .NET platform, the application vendors will use the Microsoft.Azure.KeyVault NuGet package (bit.ly/1Ji6xcS) to connect to the key vault.

Authenticating a client application with the Azure AD application is easily accomplished using the Azure AD Authentication Library (ADAL), which is also available as a NuGet package. All the application needs to authenticate with the Azure AD application is a client ID and the certificate identifier (such as the thumbprint) that can be placed in the configuration file. These can be safely deployed in the configuration file because, by themselves, they are not sensitive information.

The C# SDK provides a KeyVaultClient class, which is shown in **Figure 1**. Its constructor takes a callback method to provide the valid token from the Azure AD application. The callback method is called whenever a cryptographic action is performed using the client. A valid certificate is fetched using the custom function GetCertificateByThumbprint, using the thumbprint information from the application's configuration file, and used to authenticate against the Azure AD application. The ADAL library caches the token received from the Azure AD application the first time and serves the token from the cache for every subsequent call until the token expires.

Configuring client applications to use keys and secrets: The client application vendors now need to update the existing application to use the key and secret identifiers shared by Contoso engineers to get the required connection string and key information from the vault. Because the key identifiers by themselves are not sensitive information, they can safely be placed in the application's configuration file.

Where connection string or similar sensitive information that has been saved as a secret in the key vault is required, the existing code is replaced to read from the vault using the KeyVaultClient:

```
var connectionStringIdentifier =
ConfigurationManager.AppSettings["ConnectionStringIdentifier"];
var contosoSQLConnectionString =
await keyVaultClient.GetSecretAsync(connectionStringIdentifier);
```

First, this code looks up the connection string secret identifier that was stored in the configuration file and uses that with the SDK client to retrieve the secret value, which in this case is a connection string. The connection string can then be used as required to connect to the application database.

Where the application has to encrypt or decrypt PII information, the existing code is replaced with code to perform the operation using the SDK client and specifying the key identifier to

facebook



Microsoft
SharePoint 2010



Linked in



twitter

SEE THE WORLD AS A DATABASE

ADO.NET ▪ JDBC ▪ ODBC ▪ SQL SSIS ▪ ODATA
MYSQL ▪ EXCEL ▪ POWERSHELL

Microsoft
SQL Server

Linked in

SAP

OData
Open Data Protocol

Salesforce



facebook



Microsoft
SharePoint 2010

amazon
web services



Microsoft
Visual Studio



ODBC

Microsoft
SQL Server

Microsoft
Excel

Microsoft
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

be used. The following code shows how some text can be encrypted and then decrypted back to the original text:

```
// Within the actual application, encryption and decryption
// would happen at different parts.
var contosoPIIKeyIdentifier =
    ConfigurationManager.AppSettings["ContosoPIIKeyIdentifier"];
Byte[] textToEncrypt = Encoding.Unicode.GetBytes("Consumer PII Information");
var encryptedResult =
    await keyVaultClient.EncryptAsync(contosoPIIKeyIdentifier, "RSA_OAEP",
    textToEncrypt);
var decryptedResult =
    await keyVaultClient.DecryptAsync(contosoPIIKeyIdentifier,
    "RSA_OAEP", encryptedResult.Result);
var text = Encoding.Unicode.GetString(decryptedResult.Result);
```

Maintaining the key vault: Contoso engineers are responsible for maintaining the key vault over its lifetime, which includes changing the key value associated with a key identifier; updating the secret value associated to a secret identifier; creating new keys or secret values; updating the certificate used for Azure AD application authentication and so on.

To create or update keys and secrets in the vault, scripts like the previous ones are used. Whenever the object identifier specified for a key or a secret doesn't exist in the vault, a new object is created. For an existing identifier, the vault automatically creates a new version and makes that the current version. The older value is retained in the vault and can be used by referring to the identifier name and the version number in the URI. Client applications can decide whether to include the version number in the identifiers based on their requirements. For different application versions, you can choose either to have different vaults or identifiers with different names in the same vault, or identifiers with the same name but with explicitly specified version numbers in the identifier URI. The Contoso security policy demands a frequent change (rolling) of keys, secret values like connection strings and certificates, for which the engineers follow the following workflow:

- When keys stored in the vault are updated to newer versions, client applications that store data encrypted with those keys need to make sure the encrypted data is migrated to use the new key.
- The Contoso client application maintains the key version used to encrypt the data and always uses that when decrypting the same data.
- When encrypting the data, the application always uses new key information. This allows the application to progressively migrate to the new key values, as and when they're created.

For updating secret values, the process depends on the kind of value the secret represents. For values that provide information like connection strings, you first switch the secret's value to the standby service

Figure 2 Azure PowerShell Script for Uploading a New Certificate

```
$certificateFilePath = "C:\certificates\ContosoADApplicationNew.cer"
$x509Certificate2 =
    New-Object System.Security.Cryptography.X509Certificates.X509Certificate2
    $x509Certificate2.Import($certificateFilePath)
$rawCertData = $x509Certificate2.GetRawCertData()
$credentialValue = [System.Convert]::ToBase64String($rawCertData)
$startDate = [System.DateTime]::Now
$endDate = $startDate.AddYears(1)
$msolCredentials = get-credential
connect-msolservice -credential $msolCredentials
New-MsolServicePrincipalCredential -ServicePrincipalName
$ServicePrincipal.ApplicationId
-Type Asymmetric -Value $credentialValue -StartDate $startDate -EndDate $endDate
```

and then roll the credential information of the primary service and update the secret with the updated credential for the primary service. For certain services, like Azure Storage, this is simpler, because there are primary and secondary keys that can be used interchangeably when either is being updated. For secrets whose value is related only to the application, the secret is updated directly to reflect any new value.

When certificates used to secure the Azure AD application need to be changed, Contoso engineers use Windows PowerShell scripts to perform the update. To update the certificate for an Azure AD application, the Azure AD module needs to be installed. You'll find the module at bit.ly/10dZTIS. Contoso engineers then use the script in **Figure 2** to upload the new certificate for securing the Azure AD application.

When a certificate credential is no longer used by any application, Contoso engineers run the following script to remove the credential key from the service principal, using the `Get-MsolServicePrincipalCredential` cmdlet to get the identifier of the credential that needs to be removed:

```
$servicePrincipalCredential = Get-MsolServicePrincipalCredential
-ServicePrincipalName $ServicePrincipal.ApplicationId -ReturnKeyValues 0
Remove-MsolServicePrincipalCredential
-ServicePrincipalName $ServicePrincipal.ApplicationId
-KeysId $servicePrincipalCredential[0].KeyId
```

This script simply shows the usage of the cmdlet, by just removing the first credential from the list. In a real application, you might explicitly state the identifier of the certificate credential to be removed.

Handling multiple deployments: The application currently depends only on the Azure AD application ID, the certificate thumbprint for identifying the certificate to authenticate with the Azure AD application, and various key and secret identifiers. For different deployments of the application, different key vaults can be used. For dev deployment, each application vendor can create a key vault using their Azure subscription following the procedures I've discussed; set up the vault and populate it with the keys and secrets the application expects; and then update the configuration file. When the Contoso team needs to deploy the application for testing, they can update the configuration file with the details from the vault that's deployed under its subscription.

Wrapping Up

Contoso is happy to have made the switch to Azure Key Vault, which helped the company to easily handle most of the security risks to which its application had been subjected. It was able to accomplish everything it set out to do, with very few tweaks to the application code, and this also made it easy to manage various types of sensitive information. The ease with which Contoso was able to make this switch encourages the company to revisit its other LOB applications and also make them more secure. Azure Key Vault is something that most of the application can make use of right away, and makes storing and managing cryptographic keys and other sensitive information easy and secure. ■

RAHUL NATH is a developer, consultant and blogger with experiences ranging from building rich windows client applications to large-scale applications on the Azure platform. You can follow him on Twitter at twitter.com/raahulnath or read his blog at raahulnath.com.

THANKS to the following Microsoft technical expert for reviewing this article:
Amit Bapat

Modern Apps **LIVE!**

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in Partnership with **Magenic**

ORLANDO

ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

NOV 16-20



TECH EVENTS WITH PERSPECTIVE

5
*Great
Conferences*
1
Great Price

Leading the Modern Apps Way

Presented in partnership with Magenic, Modern Apps Live! brings Development Managers, Software Architects and Development Leads together to break down the complex landscape of mobile, cross-platform, and cloud development and learn how to architect, design and build a complete Modern Application from start to finish.

What sets Modern Apps Live! apart is the singular topic focus; sessions build on each other as the conference progresses, leaving you with a holistic understanding of modern applications, which means a complete picture of what goes into building a modern app for Windows 10, iPad, and Windows Phone devices that all interact with state-of-the-art backend services running in public or private clouds.

**REGISTER BY SEPTEMBER
16 AND SAVE \$400!**



Use promo code
MALSEP1

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



SQL Server **LIVE!**
SQL SERVER FOR MODERN DEVELOPERS

Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

SharePoint **LIVE!**
TRAINING FOR COLLABORATION

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

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

SUPPORTED BY



PRODUCED BY



MODERNAPPSLIVE.COM

Enabling DevOps on the Microsoft Stack

Micheal Learned

There's a lot of buzz around DevOps right now. An organization's custom software is critical to providing rich experiences and useful data to its business users. Rapidly delivering quality software is no longer an option, it's a requirement. Gone are the days of lengthy planning sessions and development iterations. Cloud platforms such as Microsoft Azure have removed traditional bottlenecks and helped commoditize infrastructure. Software reigns in every business as the key differentiator and factor in business outcomes. No organization, developer or IT worker can or should avoid the DevOps movement.

DevOps is defined from numerous points of view, but most often refers to removing both cultural and technology barriers between development and operations teams so software can move into production as efficiently as possible. Once software is running in production you need to ensure you can capture rich usage data and feed that data back into development teams and decision makers.

There are many technologies and tools that can help with DevOps. These tools and processes support rapid release cycles and data collection on production applications. On the Microsoft stack, tools such as Release Management to drive rapid, predictable releases and Application Insights help capture rich app usage data. This article will explore and shed some light on critical tools and techniques used in DevOps, as well as the various aspects of DevOps (as shown in **Figure 1**).

This article discusses:

- Best practices for streamlined DevOps
- Incorporating technology to automate release pipelines
- Determining a provisioning strategy

Technologies discussed:

Microsoft Azure, IIS, SQL Server, ASP.NET, Visual Studio

The Role of DevOps

Most organizations want to improve their DevOps story in the following areas:

- Automated release pipelines in which you can reliably test and release on much shorter cycles.
- Once the application is running in production, you need the ability to respond quickly to change requests and defects.
- You must capture telemetry and usage data from running production applications and leverage that for data-driven decision making versus "crystal ball" decision making.

Are there silos in your organization blocking those aspects of DevOps? These silos exist in many forms, such as differing tools, scripting languages, politics and departmental boundaries. They intend to provide separation of duties and to keep security controls and stability in production.

Despite their intentions, these silos can sometimes impede an organization from achieving many DevOps goals, such as speedy, reliable releases and handling and responding to production defects. In many cases, this silo structure generates an alarming amount of waste. Developers and operations workers have traditionally worked on different teams with different goals. Those teams spend cycles fixing issues caused by these barriers and less time focused on driving the business.

Corporate decision makers need to take a fresh look at the various boundaries to evaluate the true ROI or benefits these silos intend to provide. It's becoming clear the more you can remove those barriers, the easier it will be to implement DevOps solutions and reduce waste.

It's a challenge to maintain proper security, controls, compliance and so on while balancing agility needs. Enterprise security teams must ensure data is kept secure and private. Security is arguably as important as anything else an organization does.

However, there's an associated cost for every security boundary you build. If security boundaries are causing your teams waste and friction, those boundaries deserve a fresh look to ensure they generate ROI. You can be the most secure organization in the world, but if you can't release software on time you'll have a competitive disadvantage.

Balancing these priorities isn't a new challenge, but it's time for a fresh and honest look at the various processes and silos your organization has built. Teams should all be focused on business value over individual goals.

The Release Pipeline

The release pipeline is where your code is born with version control, then travels through various environments and is eventually released to production. Along the way, you perform automated build and testing. The pipeline should be in a state where moving changes to production is transparent, repeatable, reliable and fast. This will no doubt involve automation. The release pipeline might also include provisioning the application host environment.

Your release pipeline might not be optimized if these factors are present:

- Tool and process mismatches, whereby you have different tools and processes in place per environment. (For example, the dev teams deploy with one tool and ops deploy with another.)
- Manual steps can introduce error, so avoid them.
- Re-building just to deploy to the next environment.
- You lack traceability and have issues understanding which versions have been released.
- Release cycles are lengthy, even for hotfixes.

Provisioning

Provisioning containers is sometimes considered an optional part of a release pipeline. A classic on-premises scenario often exists in which an environment is already running to host a Web application. The IIS Web server or other host and back-end SQL Server have been running through numerous iterations. Rapid releases into these environments deploy only the application code and subsequent SQL schema and data changes needed to move the appropriate update levels. In this case, you're not provisioning fresh infrastructure (both IIS and SQL) to host the application. You're using a release pipeline that disregards provisioning and focuses only on the application code itself.

There are other scenarios in which you might want to change various container configuration settings. You might need to tweak some app pool settings in IIS. You could implement that as part of the release pipeline or handle it manually. Then you may opt to track those changes in some type of versioning system with an Infrastructure-as-Code (IaC) strategy.

There are several other scenarios in which you would want to provision as part of an automated release pipeline. For example,

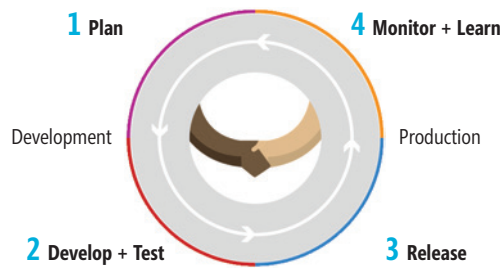


Figure 1 The Various Aspects of DevOps

early in development cycles, you might wish to tear down and rebuild new SQL databases for each release to fully and automatically test the environment.

Cloud computing platforms such as Azure let you pay only for what you need. Using automated setup and tear down can be cost-effective. By automating provisioning and environmental changes, you can avoid error and control the entire application environment. Scenarios like these make it compelling to include provisioning as part of a holistic release management system.

There are many options and techniques for including provisioning as part of your release pipeline. These will differ based on the types of applications you're hosting and where you host them. One example is hosting a classic ASP.NET Web application versus an Azure Web app or some other Platform-as-a-Service (PaaS) application such as Azure Cloud Services. The containers for those applications are different and require different tooling techniques to support the provisioning steps.

Infrastructure as Code

One popular provisioning technique is IaC. An application is an executable that can be compiled code, scripts and so on combined with an operational environment. You'll find this environment yields many benefits.

Microsoft recently had Forrester Research Inc. conduct a research study on the impact of IaC (see bit.ly/1liGRk1). The research showed IaC is a critical DevOp component. It also showed provisioning and configuration is a major point of friction for teams delivering software. You'll need to leverage automation and IaC techniques if you intend to completely fulfill your DevOps goals.

One of the traditional operational challenges is automating the ability to provide appropriate environments in which to execute applications and services, and keeping those environments in known good states. Virtualization and other automation techniques are beneficial, but still have problems keeping nodes in sync and managing configuration drift. Operations and development teams continue to struggle with different toolsets, expertise and processes.

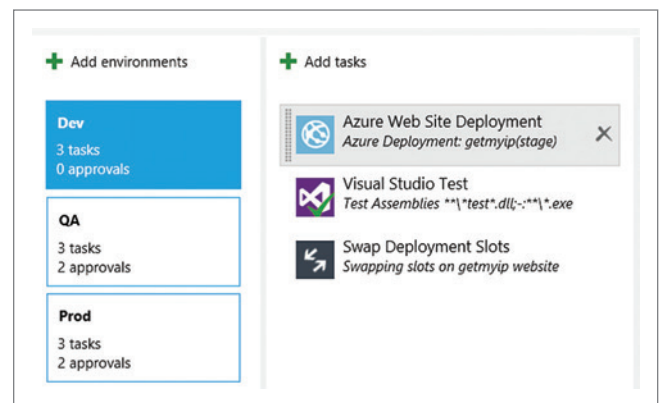


Figure 2 Tools and Options Available for Release Management

Figure 3 Separate Configuration Data Within a DCS Script

```
Configuration InstallWebSite
{
    Node $AllNodes.NodeName
    {
        WindowsFeature InstallIIS
        {
            Ensure = "Present"
            Name = "Web-Server"
        }
    }
}
InstallWebSite -ConfigurationData .\config.ps1

Where config.ps1 contains
$ConfigData = @(
    AllNodes = @(
        @(
            NodeName = "localhost"
        ))
    ))
}
```

laC is based on the premise that we should be able to describe, version, execute and test our infrastructure code via an automated release pipeline. For example, you can easily create a Windows virtual machine (VM) configured with IIS using a simple Windows PowerShell script. Operations should be able to use the same ALM tools to script, version and test the infrastructure.

Other benefits include being able to spin up and tear down known versions of your environments. You can avoid troublesome issues because of environmental differences between development and production. You can express the application environment-specific dependencies in code and carry them along in version control. In short, you can eliminate manual processes and ensure you've tested reliable automated environment containers for your applications. Development and operations can use common scripting languages and tools and achieve those efficiencies.

The application type and intended host location will dictate the tooling involved for executing your infrastructure code. There are several tools gaining popularity to support these techniques, including Desired State Configuration (DSC), Puppet, Chef and more. Each helps you achieve similar goals based on the scenario at hand.

The code piece of laC could be one of several things. It could simply be Windows PowerShell scripts that provision resources. Again, the application types and hosting environment will dictate your choices here.

For Azure, you can use Cloud Deployment Projects that leverage Azure Resource Management APIs to create and manage Azure Resource Groups. This lets you describe your environments with JSON. Azure Resource Groups also let you manage group-related resources together, such as Web sites and SQL databases. With cloud deployment projects, you can store your provisioning requirements in version control and perform Azure provisioning as part of an automated release pipeline. Here are the sections that make up the basic structure of a provisioning template:

```
{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0",
  "parameters": { },
  "variables": { },
  "resources": [ ],
  "outputs": { }
}
```

For more information on templates, go to bit.ly/1RQ3gvg, and for more on cloud deployment projects, check out bit.ly/1fIDH3m.

The scripting languages and tooling are only part of the changes needed to successfully adopt an IaC strategy. Development and operations teams must work together to integrate their work streams toward a common set of goals. This can be challenging because historically operations teams have focused on keeping environments stable and development teams are more focused on introducing new features into those environments. Sophisticated technologies are emerging, but the foundation of a successful IaC implementation will depend on the ability of the operations and development teams to effectively collaborate.

Release Orchestration

Release Management is a technology in the Visual Studio ALM stack. It's really more of a concept whereby you can orchestrate the various objects and tasks that encompass a software release. A few of these artifacts include the payload or package produced by a build system, the automated testing that happens as part of a release pipeline, approval workflows, notifications and security governance to control environments closer to production.

You can use technologies such as DSC, Windows PowerShell scripts, Azure Resource Manager, Chef, and so on to manage environment state and install software and dependencies into running environments. In terms of tooling provided by Visual Studio ALM, think of Release Management as the service that wraps around whatever technologies and tools you need to execute the deployments. Release Management might leverage simple command-line or Windows PowerShell scripts, use DSC, or even execute your own custom tools. You should aim to use the simplest solution possible to execute your releases.

It's also a good practice to rely on Windows PowerShell because it's ubiquitous. For example, you can use Windows PowerShell scripts as part of a release pipeline to deploy Azure Cloud Services.

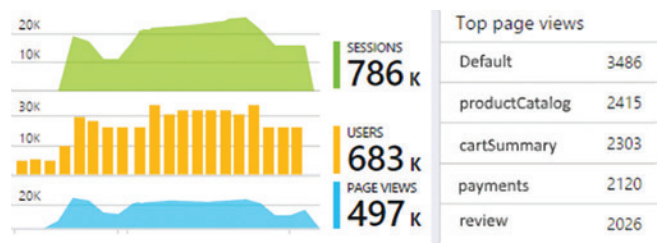


Figure 4 Application Insights Can Provide Data on Users and Page Views

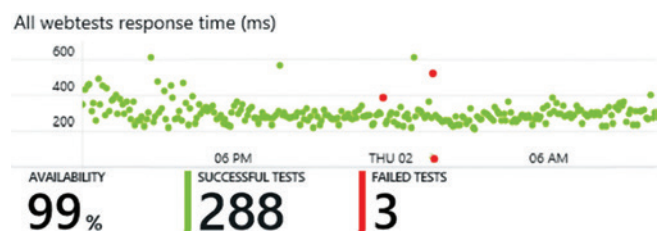


Figure 5 Application Insights Also Monitors Web Tests



Charting Collaboration

Chart the best collaboration course with SharePoint Live!, providing leading-edge knowledge and training for SharePoint administrators, developers, and planners who must customize, deploy and maintain SharePoint Server on-premises and in Office 365 to maximize the business value.

Whether you are a Manager, IT Pro, DBA, or Developer, SharePoint Live! brings together the best the industry has to offer for 5 days of workshops, keynotes, and sessions to help you work through your most pressing SharePoint projects.

**REGISTER BY SEPTEMBER
16 AND SAVE \$400!**



Use promo code
SPSEP1

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



5
*Great
Conferences*
1
Great Price

SQL Server **LIVE!**
SQL SERVER FOR MODERN DEVELOPERS

Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

SharePoint **LIVE!**
TRAINING FOR COLLABORATION

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

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

There are a lot of out-of-the-box tools with Release Management (see **Figure 2**), but you also have the flexibility to create your own.

Release Management can help you elegantly create an automated release pipeline and produce reliable automated application releases. You can also opt to include provisioning. The Release Management tooling with Visual Studio and Team Foundation Server can help you orchestrate these artifacts into the overall release transaction. It also provides rich dashboard-style views into your current and historical states. There's also rich integration with Team Foundation Server and Visual Studio Online.

Where Does DSC Fit In?

There has been a lot of press about DSC lately. DSC is not, however, some all-encompassing tool that can handle everything. You'll use DSC as one of the tools in your DevOps structure, not the only tool.

You can use DSC in pull or push modes. Then you can use the "make it so" phase to control the server state. Controlling that state can be as simple as ensuring a file or directory exists, or something more complex such as modifying the registry, stopping or starting services, or running scripts to deploy an application. You can do this repeatedly without error. You can also define your own DSC resources or leverage a large number of built-in resources.

DSC is implemented as a Local Configuration Manager (LCM), running on a target node, accepting a Management Object File (MOF) configuration file and using it to apply configuration to the node itself. So there's no hard-coupled tool. You don't even have to use Windows PowerShell to produce the MOF.

To start using DSC, simply produce the MOF file. That will eventually describe the various resources to execute, which end up written mostly in Windows PowerShell. One of the big advantages of DSC on Windows Server-based systems is the LCM is native to the OS, giving you the concept of a built-in agent. There are also scenarios for leveraging DSC with Linux. See **Figure 3** for an example of separating the configuration data for the DSC script.

DSC can be an important piece of a release pipeline if it has the resources available to help support your deployment. With on-premises or IaaS applications, DSC is an excellent choice to help control the environment configuration and support your deployment scenarios.

Still DSC isn't meant to be used for every scenario. To put this in context, if you're deploying Azure PaaS resources, it's recommended you use Azure Resource Manager to get the VMs started and the

networking configured. This isn't something DSC is designed for. Once the VMs are running, you can use DSC to get the local configuration the way you want it and ensure the configuration elements you care about don't change.

Monitor with Application Insights

Once an application and environment is in production, it's critical to collect data and monitor the operational health. You also need to understand usage patterns. This data is critical to managing a healthy service. Collecting and monitoring this data is an important piece of DevOps. For example, Microsoft has used production data to improve the Visual Studio Online teams. This rich data helps the Visual Studio Online teams ensure service availability, demonstrates to them how developers are using the service and informs decisions on feature prioritization. You can read more about the Microsoft DevOps journey at bit.ly/1AzDL9V.

Visual Studio Application Insights adds an SDK to your application and sends telemetry to the Azure Portal. It supports many different platforms and languages, including iOS, Android, ASP.NET and Java. You can capture performance data, application uptime and various usage analytics. You can show this rich data to decision makers and stakeholders to help make better decisions, detect issues and continuously improve your applications. You can read more about Application Insights at bit.ly/1lbRnrF.

Figure 4 and **Figure 5** show examples of the types of data collected by Application Insights.

Wrapping Up

DevOps helps teams drive toward continuous delivery and leverage data from running applications to help make better-informed decisions. This article has examined various prominent Microsoft technologies you can use to achieve these goals:

- Release Management lets you use any technology to drive deployments. These technologies include simple Windows PowerShell scripts, DSC configurations or even third-party tools such as Puppet.
- Infrastructure-as-Code strategies help development and operations teams efficiently work together.
- Visual Studio Application Insights gives you a mechanism to capture rich data from running applications, to help stakeholders understand application health and examine usage patterns to drive informed decision making.

These technologies can help you greatly improve your DevOps maturity. You'll also need to blend an appropriate set of technologies while working to overcome cultural barriers. ■

Additional Resources

- To learn more about Infrastructure as Code, listen to Brian Keller's discussion on Channel 9 at bit.ly/1liNqmr.
- To learn more about Azure Resource Group Deployment Projects, check out bit.ly/1fIDH3m.
- To learn more about TFS Planning, Disaster Avoidance and Recovery, and TFS on Azure IaaS, check out the guide at vsarplanningguide.codeplex.com.
- To learn more about Config as Code for DevOps and ALM practitioners, check out vsardevops.codeplex.com.

MICHAEL LEARNED is a Visual Studio ALM Ranger currently focused on DevOps and Microsoft Azure. He has worked on numerous software projects inside and outside of Microsoft for more than 15 years. He lives in central Illinois and devotes his free time to helping the community, as well as relaxing with his daughter, two sons and wife. Reach him on Twitter at twitter.com/mlhoop.

THANKS to the following technical experts for reviewing this article: Donovan Brown (Microsoft), Wouter de Kort (Independent Developer), Marcus Fernandez (Microsoft), Richard Hundhausen (Accentient), Willy-Peter Schaub (Microsoft) and Giulio Vian (Independent Developer)

SQL Server[™] **LIVE!**

TRAINING FOR DBAs AND IT PROS

ORLANDO
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

**NOV
16-20**



**LIVE!
360**
TECH EVENTS WITH PERSPECTIVE

5
*Great
Conferences*
1
Great Price

Driving Data Forward

After 5 days of workshops, deep dives and breakout sessions, SQL Server Live! will leave you with the skills needed to Drive Data Forward.

With timely, relevant content, SQL Server Live! helps administrators, DBAs, and developers do more with their SQL Server investment. Sessions will cover performance tuning, security, reporting, data integration, adopting new techniques, improving old approaches, and modernizing the SQL Server infrastructure.

**REGISTER BY SEPTEMBER
16 AND SAVE \$400!**



Use promo code
SQLSEP1

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



SQL Server[™] **LIVE!**
TRAINING FOR DBAs AND IT PROS

Visual Studio[™] **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

SharePoint[™] **LIVE!**
TRAINING FOR COLLABORATION

Modern Apps[™] **LIVE!**
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

SUPPORTED BY



msdn
MAGAZINE

Redmond
MAGAZINE

PRODUCED BY



SQLLIVE360.COM



Computing with Artificial Spiking Neurons

A fascinating area of computer science is computing with artificial spiking neurons—small software components that model the behavior of biological neurons. Artificial spiking neurons are related to but quite different from the artificial neurons in a common software neural network.

Let me state right up front that this discussion of artificial spiking neurons is not likely to be immediately useful to you in your normal day-to-day programming tasks. But it will give you insight into what the future of computing might be like, and you just might find artificial spiking neurons interesting in their own right.

The best way to get a feel for what a spiking neuron is and to see where this article is headed is to take a look at the demo program in **Figure 1**.

The demo shows the input and output values for a single spiking neuron. There are three input streams, sometimes called spike trains.

Artificial leaky
integrate-and-fire spiking
neurons are extremely simple.

Each of the input spike trains has 16 0 or 1 values. The three input spike trains are:

```
0010101111001011
1011000111011011
1101000110111111
```

These 0 or 1 values represent input over time from some other spiking neuron. In other words, at time $t = 0$, the input value from the first spike train is 0; at time $t = 1$, the input is 0; at $t = 2$, the input is 1; and so on through $t = 15$ when the input value is 1.

At time $t = 0$, the spiking neuron receives input from all three streams, so at $t = 0$, the complete input to the neuron is (0, 1, 1); at time $t = 1$, the input is (0, 0, 1); and so on through $t = 15$ when the input is (1, 1, 1).

The next part of the demo program displays numeric constants that define the behavior of the neuron. There are three weights with values (4, -2, 3) that correspond to each input train. The meaning of the leak potential (1), the threshold potential (8), the spike

potential (4), and the post-spike latency (2) will be explained shortly. In the context of spiking neurons, the term “potential” means (loosely) electrical voltage rather than “possible” or “impending.”

The demo program simulates 16 time ticks. At each tick, the three input 0 or 1 values are displayed along with the state of the neuron (active or inactive), and the current electrical potential (V) of the neuron. Notice that at time $t = 6$ the neuron reaches a potential of $V = 8$ and spikes to $V = 12$. Behind the scenes, as each set of input values is processed, an output value of 0 or 1 is generated and saved

```
file:///C:/SpikingNeuron/bin/Debug/SpikingNeuro...
Begin spiking neuron demo
The inputs are:
0 0 1 0 1 0 1 1 1 1 0 0 1 0 1 1
1 0 1 1 0 0 0 1 1 1 0 1 1 0 1 1
1 1 0 1 0 0 0 1 1 0 1 1 1 1 1 1

The weights are: +4 -2 +3

The leak potential is:      1
The threshold potential is: 8
The spike potential is:    4
The post-spike latency time is: 2

Starting processing

t = 0. Inputs = 0 1 1. Neuron is active. V = 0
t = 1. Inputs = 0 0 1. Neuron is active. V = 2
t = 2. Inputs = 1 1 0. Neuron is active. V = 3
t = 3. Inputs = 0 1 1. Neuron is active. V = 3
t = 4. Inputs = 1 0 0. Neuron is active. V = 6
t = 5. Inputs = 0 0 0. Neuron is active. V = 5
t = 6. Inputs = 1 0 0. Neuron is active. V = 8
Spiking, V = 12
t = 7. Inputs = 1 1 1. Neuron is inactive. V = 0
t = 8. Inputs = 1 1 1. Neuron is inactive. V = 0
t = 9. Inputs = 1 1 0. Neuron is active. V = 1
t = 10. Inputs = 0 0 1. Neuron is active. V = 3
t = 11. Inputs = 0 1 1. Neuron is active. V = 3
t = 12. Inputs = 1 1 1. Neuron is active. V = 7
t = 13. Inputs = 0 0 1. Neuron is active. V = 9
Spiking, V = 13
t = 14. Inputs = 1 1 1. Neuron is inactive. V = 0
t = 15. Inputs = 1 1 1. Neuron is inactive. V = 0

Output spike train =
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

End spiking neuron demo
```

Figure 1 Artificial Spiking Neuron Demo

Code download available at msdn.microsoft.com/magazine/msdnmag0915.

in a buffer. After all the input values have been processed, the demo program displays the resulting output spike train:

0 0 0 0 0 1 0 0 0 0 0 1 0 0

At this point, you're probably not too impressed. A bunch of 0 or 1 values goes in, and a bunch of 0 or 1 values comes out. But bear with me and you'll see why artificial spiking neurons may help lead to a fundamental change in computers and computing.

Artificial spiking neurons can accommodate any number of streams.

This article assumes you have at least beginner programming skills but doesn't assume you know anything about spiking neurons. I've coded the demo program using C#, but you shouldn't have much difficulty refactoring it to another language, such as Python or JavaScript. The demo code is short and is presented in its entirety in this article. The complete source is also available in the accompanying code download.

Understanding Spiking Neurons

Take a look at the graph in **Figure 2**. The graph displays the electrical potential value, V , of the demo spiking neuron from time $t = 0$ to $t = 15$. There are many different variations of spiking neurons. The demo program is based on a model called the leaky integrate-and-fire spiking neuron.

The electrical potential of the neuron tends to increase over time. When V meets or exceeds the spiking threshold equaling 8, indicated by the dashed line, the V value instantly spikes up by 4 (the spike potential) and then immediately resets to 0. Spiking events occur at time $t = 6$ and $t = 13$. If a spiking event occurs, a 1 is emitted to the output spike train, otherwise a 0 is emitted.

So, just how is V calculated for each value of t ? At time $t = 0$, the three input values are (0, 1, 1). Recall the neuron's three weight values are (4, -2, 3). First, the sum of the products of inputs times weights is computed and added to the current value of V . If V is assumed to be 0 at the start of the demo then:

$$\begin{aligned} V &= V + \text{sum} \\ &= 0 + (0)(4) + (1)(-2) + (1)(3) \\ &= 0 + 1 \\ &= 1 \end{aligned}$$

This is the integrate step. Next, the leak value is subtracted. For the demo, $\text{leak} = 1$ so:

$$\begin{aligned} V &= V - \text{leak} \\ &= 1 - 1 \\ &= 0 \end{aligned}$$

The new value of $V = 0$ doesn't exceed the threshold value of 8 so the neuron doesn't spike and it emits a 0 to the output spike train. Next, at time $t = 1$, the input values are (0, 0, 1). Combining the integrate and leak steps gives:

$$\begin{aligned} V &= V + \text{sum} - \text{leak} \\ &= 0 + (0)(4) + (0)(-2) + (1)(3) - 1 \\ &= 0 + 3 - 1 \\ &= 2 \end{aligned}$$

Artificial leaky integrate-and-fire spiking neurons are extremely simple. Notice that all values are integers, which, combined with design simplicity, mean that artificial neurons can be efficiently implemented in software or hardware.

In **Figure 2**, you can see that a spike event occurs at time $t = 6$. At $t = 5$, $V = 5$ so the integrate-and-leak calculation for $t = 6$ is:

$$\begin{aligned} V &= V + \text{sum} - \text{leak} \\ &= 5 + (1)(4) + (0)(-2) + (0)(3) - 1 \\ &= 5 + 4 - 1 \\ &= 8 \end{aligned}$$

At this point, V meets the threshold value of 8, so the value of V spikes to $8 + 4 = 12$, an output value of 1 is emitted, and then V is immediately reset to 0. After a spike event, the simulated neuron enters a state of inactivity where input values are ignored. The demo sets this latency period to 2, so at times $t = 7$ and $t = 8$, V remains at 0 regardless of the values of the inputs. At time $t = 9$, the neuron becomes active and resumes normal behavior.

Implementing the Demo Program

The code for the demo program, with a few minor edits to save space, is presented in **Figure 3**. To create the demo program, I launched Visual Studio and created a new C# console application project named SpikingNeuron. The demo program has no significant Microsoft .NET Framework dependencies so any relatively recent version of Visual Studio will work.

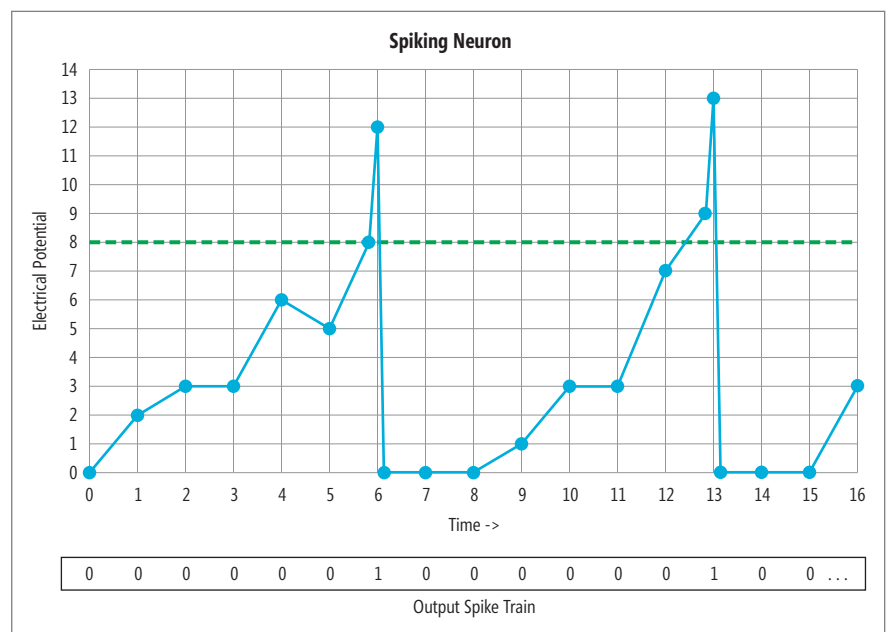


Figure 2 Leaky Integrate-and-Fire Spiking Neuron Behavior

After the template code loaded into the text editor, in the Solution Explorer window I renamed file Program.cs to SpikingNeuronProgram.cs and allowed Visual Studio to rename class Program for me. At the top of the source code I removed all unneeded using statements, leaving just the reference to the top-level System namespace.

The code in the Main method begins by setting up the input data and a storage buffer for the output values:

```
int[][] inputs = new int[3][];
inputs[0] = new int[] { 0, 0, 1, 0, 1, 0, 1, 1,
    1, 1, 0, 0, 1, 0, 1, 1 };
...
int[] output = new int[16];
```

Here there are three input streams. Artificial spiking neurons can accommodate any number of streams. Each stream has 16 0 or 1 values. The input length is arbitrary, but from a programmer's point of view, you can think of each demo input stream as a 16-bit unsigned value.

Next, the demo sets values that define the neuron's behavior:

```
int[] wts = new int[] { 4, -2, 3 };
int leak = 1;
int v = 0; // Electrical potential (voltage)
int thresh = 8; // Needed to fire an output spike
int spike = 4; // Increase in v at spike event
int tNext = 0; // Next time when neuron is active
int latency = 2; // Number t inactive after spike
```

An inactive neuron is essentially asleep, so there's no change to the electrical potential and no spike can occur.

Figure 3 Spiking Neuron Program

```
using System;
namespace SpikingNeuron
{
    class SpikingNeuronProgram
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Begin spiking neuron demo");

            int[][] inputs = new int[3][];
            inputs[0] = new int[] { 0, 0, 1, 0, 1, 0, 1, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 1 };
            inputs[1] = new int[] { 1, 0, 1, 1, 0, 0, 0, 1,
                1, 1, 0, 1, 1, 0, 1, 1 };
            inputs[2] = new int[] { 1, 1, 0, 1, 0, 0, 0, 1,
                1, 0, 1, 1, 1, 1, 1, 1 };

            Console.WriteLine("The inputs are: ");
            for (int i = 0; i < inputs.Length; ++i)
                ShowVector(inputs[i], false);
            Console.WriteLine("");

            int[] output = new int[16];

            int[] wts = new int[] { 4, -2, 3 };
            Console.WriteLine("The weights are: ");
            ShowVector(wts, true);
            Console.WriteLine("");

            int leak = 1;
            Console.WriteLine("Leak potential is: " + leak);

            int v = 0; // electrical potential (voltage)
            int thresh = 8; // Threshold
            int spike = 4; // Increase in v at spike
            int tNext = 0; // Time when neuron is active
            int latency = 2; // Inactive after spike

            Console.WriteLine("Threshold is: " + thresh);
            Console.WriteLine("Spike is: " + spike);
            Console.WriteLine("Latency time is: " + latency);
            Console.WriteLine("Starting processing\n");

            for (int t = 0; t < 16; ++t)
            {
                Console.WriteLine("-----");
                Console.WriteLine("t = ");
                Console.WriteLine(t);
                if (t <= 9) Console.WriteLine(" ");
                Console.WriteLine(t + ". ");

                Console.WriteLine("Inputs = " + inputs[0][t] +
                    " " + inputs[1][t] +
                    " " + inputs[2][t]);

                if (t != tNext) // Neuron not active
                {
                    Console.WriteLine(". Neuron is inactive. ");
                    Console.WriteLine("V = " + v);
                    output[t] = 0;
                }
                else // Neuron is active
                {
                    Console.WriteLine(". Neuron is active. ");

                    int sum = 0;
                    for (int j = 0; j < inputs.Length; ++j)
                        sum += inputs[j][t] * wts[j];
                    v = v + sum;

                    v = v - leak;
                    if (v < 0)
                        v = 0;

                    Console.WriteLine("V = " + v);

                    if (v >= thresh) // Spike and reset
                    {
                        v = v + spike;
                        Console.WriteLine(" Spiking, V = " + v);
                        output[t] = 1;
                        v = 0;
                        tNext = t + 1 + latency;
                    }
                    else
                    {
                        output[t] = 0;
                        tNext = t + 1;
                    }
                }
            } // Active
        } // t
        Console.WriteLine("-----");
        Console.WriteLine("Output spike train = ");
        ShowVector(output, false);

        Console.WriteLine("End spiking neuron demo");
        Console.ReadLine();
    } // Main

    static void ShowVector(int[] vector, bool plus)
    {
        for (int i = 0; i < vector.Length; ++i)
        {
            if (plus == true && vector[i] >= 0)
                Console.WriteLine("+");
            Console.WriteLine(vector[i] + " ");
        }
        Console.WriteLine("");
    }
} // Program
} // ns
```

Notice that the weights for input streams 0 and 2 are positive and therefore act to increase the electrical potential of the neuron, but the weight for input stream 1 is negative, so it decreases the potential. Sometimes these are distinguished as excitatory (increase) and inhibitory (decrease) inputs.

Here the leak value is set to 1. An alternative to consider is to make the leakage stochastic; that is, randomly vary the leak value between, say, 0 and 3 at each time tick. Or you could vary the leak value to be proportional to the current potential. You can also consider making the spike reset latency time period a random value.

All the processing is controlled by a time-driven loop:

```
for (int t = 0; t < 16; ++t)
{
    // Compute new V
    // Spike if V >= threshold
    // Emit a 0 or 1
}
```

Inside the time loop, the demo first checks to see if the neuron is inactive:

```
if (t != tNext) // Neuron is not active
{
    Console.WriteLine(". Neuron is inactive. ");
    Console.WriteLine("V = " + v);
    output[t] = 0;
}
else
{
    // Active
}
```

Variable `tNext` is the next time value when the neuron will be active. In most cases, `tNext` will be `t+1`. An inactive neuron is essentially asleep, so there's no change to the electrical potential and no spike can occur. Inside the active-neuron branch, the electrical potential `V` is calculated:

```
int sum = 0;
for (int j = 0; j < inputs.Length; ++j)
    sum += inputs[j][t] * wts[j];
v = v + sum;

v = v - leak;
if (v < 0) v = 0;
```

Here `j` is the index of the input stream (0, 1, 2) and `t` is the time index. For example, `inputs[1][8]` is the 0-or-1 value for input stream 1 at time `t = 8`. After the leak is subtracted from `V`, the resulting value is checked to make sure `V` doesn't go negative. However, in real neurons, electrical potentials can in fact be negative, so an option to consider is to allow `V` to go negative.

After `V` is calculated, its value is checked to see if a spike event should occur, as shown in **Figure 4**.

When a spike event occurs, the current value of `V` is incremented (by 4 in the demo) and then immediately reset to 0. In other words,

Figure 4 Checking If a Spike Event Should Occur

```
if (v >= thresh) // Spike and reset
{
    v = v + spike;
    Console.WriteLine(" Spiking, V = " + v);
    output[t] = 1; // Fire
    v = 0;
    tNext = t + 1 + latency;
}
else // No spike
{
    output[t] = 0;
    tNext = t + 1;
}
```

the temporary spiked value of `V` is not used at all. An alternative for you to consider is to not automatically reset to `V = 0`. Instead, you can reset by subtracting some amount from the spiked `V` value. For example, at time `t = 6` in the demo, `V` temporarily spikes from 8 to 12. If you used a spike reset value of 10, then instead of resetting from 12 to 0, the neuron would reset from 12 to 2.

What's the Point?

Artificial spiking neurons are studied by different groups of people for different purposes. Neurobiologists attempt to create software models that exactly replicate the behavior of real neurons in order to gain insights into biological processes. It turns out that the leaky integrate-and-spike neuron model is too simple to completely replicate the behavior of real neurons, so more complex models are usually used.

Although artificial neural networks that use real-valued simulated neurons have been studied for decades, networks of artificial spiking neurons haven't been explored much.

Artificial spiking neurons also serve as the basis for machine learning classification systems. Although artificial neural networks that use real-valued simulated neurons have been studied for decades, networks of artificial spiking neurons haven't been explored much. In my opinion, research results in this area are inconclusive and it's not clear whether networks of artificial spiking neurons provide any advantage over classical real-valued artificial neurons. There are still many open research questions.

Finally, artificial spiking neurons are being used in an effort to create a fundamentally new approach to computers and programming. The U.S. Defense Advanced Research Projects Agency (DARPA) Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE) project aims to create computers that can scale to biological levels. Instead of using traditional hardware architecture, some initial promising designs have used billions of inter-connected artificial leaky integrate-and-fire spiking neurons. If these efforts prove successful, it may be possible to create computers that are almost inconceivably more powerful than current systems. ■

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 technical experts for reviewing this article: Todd Bello and Dan Liebling



How To Be MEAN: Node.js

Microsoft has been adopting technology from across the software spectrum as part of its rebranding and “re-relevancing.” One of the technologies it has adopted is Node.js. This gives developers a golden opportunity to use one of the more popular full-stack software groupings on the Node.js platform known as MEAN: MongoDB, Express, AngularJS and Node.js.

In the last installment (msdn.microsoft.com/magazine/mt185576), I got the basic Node.js parts up and running. In this one, I’ll spin up a simple Node “Hello World” Web endpoint, and deploy it to a Microsoft Azure Web site. Over the next several installments, I’ll slowly build my way through the MEAN stack, working from the ground up.

As discussed in the previous article, there are a lot of places where I could swap out parts of the MEAN stack for something else—DocumentDB for MongoDB, ASP.NET WebAPI for Node.js and ASP.NET MVC for Express, or BackboneJS (or any of a whole host of other JavaScript Single-Page-Application frameworks) for AngularJS—but none of the alternatives enjoys the popularity that MEAN currently holds (at least among JavaScript aficionados).

Node.js

Node.js is fundamentally just “JavaScript on the server.” Yes, there’s a different programming approach to Node.js that tries to deal with concurrent execution. Instead of block on a call, the programmer passes in a function literal to be invoked when the operation is completed. This lets the developer to think of code as single-threaded, even though multiple threads are in use below the surface. On the whole, though, the largest distinction of Node.js is you use JavaScript to build the server components, instead of C#, Java or Ruby. In that sense, it’s really just a change of scenery—not an entirely different universe.

Node.js is fundamentally just
“JavaScript on the server.”

The simplest Node.js application is, of course, the ubiquitous “Hello, World,” which you can easily write using the built-in console object:

```
console.log("Howdy, NodeJS!");
```

Assuming this goes into a file called `hello.js` in the current directory, you’d run it using the `node` utility at the command-line using `node hello.js`. Or you could let Node.js infer the filename extension by simply running “`node hello`.” Either way, Node greets you in the traditional way.

Like most programming platforms, Node.js has its own set of libraries and APIs out of the box. As I pointed out last time, Node.js uses the `require` convention to reference an installed library. This traps the returned object into a local variable of the same name. So, for example, if I want to write a simple HTTP server that will effectively give me the same greeting over the HTTP protocol, I can put the following into a simple `helloHTTP.js` file:

```
var http = require('http');
var port = process.env.PORT || 3000;
http.createServer(function(req, res) {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World\n');
}).listen(port);
```

The `require` line finds the “`http`” library within the Node.js installation, and stores it to the `http` object in dependency injection. This is a standard Node.js convention, and should be held as fairly sacrosanct. The second line uses the built-in `process` object to access the surrounding environment. In this case, it uses the “`env`” object within the `process` object to determine whether an environment variable named `PORT` is set to anything. If it is, I’ll use that as the port on which to run the server. Otherwise, I’ll use the default port 3000. Many Node.js frameworks prefer port 3000 as the default, for obscure historical and cultural reasons.

The nature of Node.js programming becomes clearer in the next few lines. I use the `http` object to create an HTTP server. The sole parameter is a function literal that takes a `req` (HTTP request) object and a `res` (HTTP response) object as parameters, and uses `res` to write the HTTP response back. This idiom is ubiquitous throughout all levels of the Node.js stack.

This is one of those “you love it or you hate it” kinds of things. You’ll be seeing more of this in the articles to come. So if this isn’t clear, spend some time experimenting. The returned object from `createServer` is then bound to the desired port using the `listen` call. Lo and behold, you have a running HTTP server you can easily run using “`node helloHTTP`” and pointing a browser at `http://localhost:3000`.

Azure Command-Line Tools

From my last article, you’ll recall Node.js has a package utility called the Node package manager (`npm`) you can use to download dependent libraries. You can also use it to download tools you can then use from the command line. This is a subtle, but powerful, aspect of Node.js. It acts as a platform-neutral “playing field equalizer.”

You can effectively hide any distinctions between Windows, Mac OS or Linux behind a wall of JavaScript scripts. Microsoft picked up on this a while back, and packaged a set of command-line tools into a Node.js package called `azure-cli`. Installing it is easy with `npm`:

```
npm install -g azure-cli
```


The `-g` flag tells npm to install the tools “globally” (meaning they’re not tied to the local directory in which the command is run). This makes the resulting package available for use across the entire system. When it’s finished, a new command-line utility, `azure`, will be available for use. The `azure-cli` package doesn’t give any greater or lesser functionality than using the Azure Portal. The `azure-cli` tool’s advantage lies in its ability to script `azure` commands as part of an automated deploy system, for example.

Deploy to Azure

So, if you want this lovely little greeting to be available to the world over the Internet, you need to create an Azure Web Site as a host. Using the “`azure`” tool, it’s pretty simple. First, you have to tie the tool in to the account in Azure:

```
azure account download
```

This will fire up the system’s default browser pointing to the Azure login portal. Use the Azure account credentials to log in. When you’re done, it will automatically download a `publishSettings` file containing the credential information the `azure` tool needs, which can be directly imported:

```
azure account import <filename>
```

`Filename` will often be something like “Visual Studio Ultimate with MSDN-4-23-2015-credentials.publishsettings,” depending on the Azure subscription details and the date you download the `publishSettings` file. Once that’s done, it’s a simple matter to create a new Azure Web site by setting up Git to do deployments to the site:

```
azure site create -git
```

You can effectively hide any distinctions between Windows, Mac OS or Linux behind a wall of JavaScript scripts.

This will prompt for a site name, and (assuming that name is unique) spin up the appropriate Web site. If all that works, spin up the current directory as a git-initialized local repository. Assuming you’re still in the same directory that holds the `helloHTTP.js` file from before, you can add it to the Git repository and push it to the Azure cloud:

```
git add helloHTTP.js
git commit -m "Initial commit"
git push azure master
```

Git will think about it for a few seconds. Then it will go through a sequence of steps that remain opaque for the moment. When it’s finished, though, Azure will hold the new Node.js code, and you can browse to it, as shown in **Figure 1**.

This is why the `helloHTTP` code uses either the default port 3000 or the environment variable `PORT` from the surrounding process. On the Azure cloud, `PORT` is set to a value the Azure infrastructure

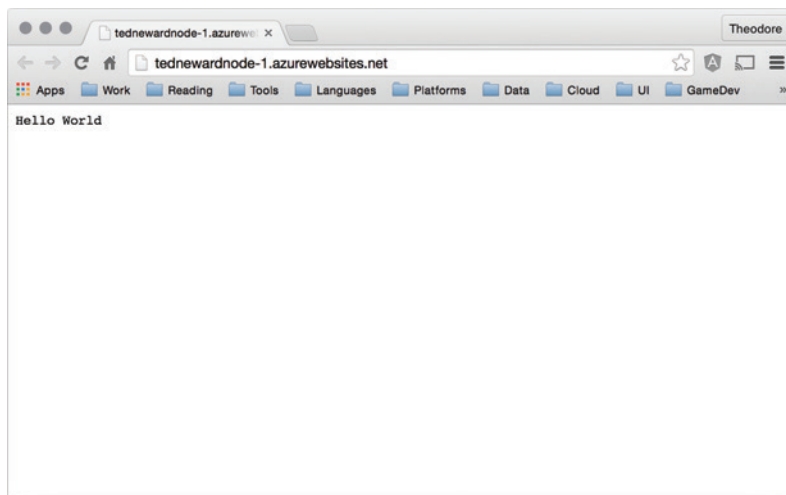


Figure 1 Hello, Node World

maintains. This is so Microsoft can manage the various service endpoints more efficiently.

That’s really the last of the installation steps you need to do. Honestly, it takes longer to read about it than it does to actually run the commands, once you’re past the initial setup stages. And even more honestly, any Azure work or exploration will require similar kinds of setup. Azure really represents a next-step platform for many development efforts. In other words, this is something you’ll need to know how to do eventually, so you may as well learn about it now.

From here, pretty much everything will be MEAN-related. The only time you’ll need to bring Azure details to mind is when you need to configure the environment to point to the MongoDB database server, for example, or when dealing with how Azure interacts with Node.js.

Wrapping Up

It’s worth pointing out that Node.js isn’t just an HTTP pipeline. In fact, Node can run any kind of networked application by opening up the right library. The same is true of the Microsoft .NET Framework. However, like the .NET Framework, most Node.js applications are going to be HTTP-based in nature.

The `http` library in Node.js is pretty low-level. As a result, the Node.js community developed a higher-level library and set of abstractions to make dealing with HTTP-based endpoints easier. This library is called Express. It lets you build what the Node.js community calls “middleware.” That’s what I’ll be looking at next time.

For now, experiment with the Node.js `http` library, but don’t get too attached, because I’ll be leaving it behind pretty quickly once I get into the next iteration. In the meantime, however ...

Happy coding. ■

TED NEWARD is the CTO at iTrellis, a consulting services company. He has written more than 100 articles and authored or co-authored a dozen books, including “Professional F# 2.0” (Wrox, 2010). He is an F# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com or ted@itrellis.com if you’re interested.

THANKS to the following technical expert for reviewing this article:
Shawn Wildermuth



Usability Practices for Modern Apps

Usability is an important but often overlooked aspect of application development. Developers tend to leave usability up to the UX experts and UI designers. However, there are some usability tricks—some small, some larger—developers can apply that have a big impact.

The primary motivator is developing software that's actually easy to use. This means popular commands are prominent and readily available. Less popular commands are available with as few clicks as possible. I'll review several usability tricks that will help you build great modern apps.

Many developers dislike the term “best practices.” It seems to imply the practice in question is the only way to do something, and you know that's just not true. There are practices that work well in most scenarios, and a broad spectrum of techniques that may or may not work well. That's why most are reluctant to use the term “best practices.” Instead I'll use the term “good practices.” These practices generally work as well as, if not better than, other ways of doing things. That doesn't mean you should apply them in every scenario, though.

Bad practices are often clearly bad. They're design or development techniques for which there is a better way to do things. It's easy to spot bad practices as they manifest themselves in frustrating software. Any time you've ever wanted to pull your hair out because you couldn't use software, a bad practice was the culprit.

Many developers dislike the term
“best practices.”

Good Usability Practices

The first and arguably most important usability practice is accessibility. The more accessible the software, the easier it is to use. There are a few types of users for whom accessibility is a challenge. You must consider those with low or no hearing, visual impairment, motor impairment and cognitive impairment. It's relatively easy to develop Web sites and apps that incorporate accessible design. Many of those techniques include small modifications to HTML or CSS. See my column, “Design and Develop Accessible Modern Apps” (msdn.microsoft.com/magazine/dn913189), for more details on accessible design techniques.

Popular commands should be prominently displayed. Don't make your users dig for information. Instead, use something like the Visual Studio Quick Launch functionality that lets users search for commands by typing the name or a synonym of the command.

Add a Quick Launch feature to the regular menus and navigation. Along the same lines, don't make users work too hard to perform tasks. Many users will stop at any roadblocks, and review apps in app stores poorly when they can't easily accomplish tasks.

The more accessible the
software, the easier it is to use.

Consistency is the key to building great software. Use consistent aesthetics, within the app itself and between the app and its OS. Anyone can tell you how frustrating it is to try to use an app on an Android or Windows device that tries to look and behave like an iPhone app. Part of the reason people buy devices is because they like its style. Stick with the style paradigms of the host OS, and then add your own flair. This means you'll need separate code bases, but you can still share back-end code between the apps.

Consistent navigation is also essential. Use the same navigation style throughout the Web site or app. As part of good navigational design, make sure users have quick access to their frequently used commands. You can determine those commands by logging every time the user launches them. Ensure that elements such as buttons and controls in forms have consistent labels. If you're writing a novel, getting out the thesaurus and mixing synonyms improves your prose. In software, using alternate words or phrases for the same actions or fields only confuses users.

Everybody already has too many logins. As a side effect of that, many use the same password for all their resources. This is the kind of thing security experts warn you not to do. Then developers design Web sites that encourage this bad practice by insisting you code your own security infrastructure instead of using a trusted security provider such as OAuth, Microsoft or Google. Even Facebook or Twitter authentication is better than creating your own.

Whenever possible, give your users options. You can create your own login system, but then you must maintain it and you'll be responsible if data is ever breached. Going with a trusted provider shifts the responsibility to them. They have entire teams working on maintaining and securing user data. Leaving the security to security experts leaves you time to be the business expert and solve the business problems with your software.

Proper defaults go a long way toward accessibility. It's 2015, yet many Web sites and apps still force you to enter a city and state,

AWSInsider.net

Your independent resource for news, how-tos, tips and more on the Amazon Web Services cloud platform.



instead of asking for a ZIP code and automatically filling in the details. Entering a small sequence of numbers is quite easy for users as compared to having to select from a massive list of items in a small dropdown. Check out my column, “A Mobile-First Approach to Modern App Development” (msdn.microsoft.com/magazine/dn948114), for more details on building software using a mobile-first design.

Responsive UIs scale up and down to fit the form factor and capabilities of a specific device. Obviously, tailoring software to the device is a good thing for users. Responsive UIs tend to be mobile-first. And mobile-first designs tend to be more useable than traditional software. For more on building responsive UIs, check out “Build a Responsive and Modern UI with CSS for WinJS Apps” (msdn.microsoft.com/magazine/dn451447).

Consistency is the key to building great software.

A minimalist design is another hallmark of modern apps. Users will dump an app that overwhelms them with lots of rarely used choices. They’ll use apps that are straightforward and present choices as contextually necessary. In apps for Windows 8 and later, you can use app bars that can hide while the user works in the app, then present options as needed.

Bad Usability Practices

Many Web sites and apps include forms requiring the user to enter his e-mail twice. If he can’t get his e-mail correct, why does anyone think he’ll enter his address, city, state or other information correctly? Why aren’t all those fields duplicated?

Asking a user to re-enter his e-mail address is irritating. That’s because humans are creatures of habit. Typing and typos are like other habits. A user is just as likely to make the same typos in both boxes.

The concept of a Captcha is noble. A Web site or app will show an image containing a number, word or phrase. Then the user enters that number, word or phrase. The number, word or phrase in the image is usually obscured somewhat so computers can’t identify the image. The problem is nowadays AI systems can perform image recognition and easily recognize a Captcha. Most humans ironically have a difficult time. Human eyes are about par with computers when looking at an obscured pattern. Sometimes what seems like a good idea may not be after all.

Infinite scrolling and automatic page refreshes are also problematic, especially for those who rely on assistive technologies such as screen narrators. While Facebook, and more recently Twitter, do infinite scrolling well, there can still be difficulties when new content arrives and pushes old content aside. Consider trying alternative input and accessible devices if you’re implementing these features.

Tiny close buttons can drive even skilled users crazy. Often, people with touch devices can’t even use them because the tiny area is non-responsive to a tap. It’s not as bad if there’s a prominent close button. However, many dialog boxes contain only a button to continue, but no way to back out. Responsive modern apps always provide easy ways to back out of actions.

Bad formatting is the bane of good Web site design. Many users cringe when they see a telephone field in a form. You never know what kind of crazy validation is happening. Does this field want the country code? Should I enter the parenthesis and dashes with the number, or just enter the number? Users will often try to enter a number they believe to be in the correct format, only to be promptly scolded by a modal dialog that further interrupts their workflow.

Along with scary phone fields come URL fields. Many of these won’t automatically put in the `http://`. These same Web sites and apps never tell you the `http://` is required until you click submit, fail validation and the form fields reset themselves.

There’s an entire set of bad practices concerning form failures. The entire reason forms exist is to capture data. Yet developers often make forms difficult and confusing. It’s maddening to attempt to fill out a form that insists on clearing its data if you forget something or do something wrong. Many Web sites and apps continue to do this. It takes twice the time to get things done.

If your app is trying to make money, this is a bad way to go. You’re blocking the user from making purchases. The more likely an app is to do things like clearing forms is proportional to how long it takes to fill it out. The longer it takes, the more likely a developer has decided to clear the form on unsuspecting users.

Another form fail is not implementing proper defaults. Are your users mostly from your country? Then default to that, but show the other countries as well. The same thing applies to states, ZIP codes, or the most popular value of any particular field. If your app has already been deployed, you can check the database for those values. If still in development, ask your users. One default Web sites never seem to miss, though, is setting the option to receive a newsletter or product information to checked.

Many Web sites and apps don’t provide proper search functionality. Some don’t provide search functionality at all. Search Engine Watch says a whopping 92 percent of Web users use at least one search engine all the time.

Users will dump an app that overwhelms them with lots of rarely used choices.

Present search results in easy-to-consume summaries and in a clear and consistent manner. Put ads around the search results or content, but not on top of it. When a user sees an ad inside a set of search results, she might think that’s the end of the results, so she’ll leave because she didn’t find anything.

There are several aesthetic don’ts in UI design. I like to call them “UI eyesores.” This includes things like tiny fonts that need to be zoomed in on to be read, too many ads or ads that contain ads. Many Web sites and apps rely on advertising revenue, so ads are a necessary evil. However, if your ads contain ads (yes, that is happening), it’s a disaster.

Another popular usability anti-pattern is button colors and placement. Often, a button designated to take or continue with an action is colored bright red. This color indicates stopping or

canceled an action. Like traffic lights, green, yellow, and red colors tend to indicate go, caution, and stop, respectively.

Other Usability Practices

Navigation goes under “other usability principles” because there are some types of navigation that are conducive to ease of use, and some that aren’t. When designing a navigation scheme, it should be clear and consistent. It’s usually best to go with the host OS paradigms. Maintaining that consistency between the app and the OS makes it easier on users. This is especially true of new users or those who don’t use much technology. Nevertheless, it still holds true for experienced users who like shortcuts based on paradigm consistency.

Tabbed menus are often more appropriate and easier for navigation. However, if there are too many rows of tabs, it becomes difficult to navigate. Desktop apps do well with traditional pulldown menu structures, but Web apps and native device apps often require a different scheme. For example, Windows Phone uses Live Tiles to enhance the user’s experience by providing large, easy-to-tap areas to launch apps or navigate.

Hamburger menus are said to significantly drive down app or Web site engagement by a significant amount, according to the technology media site TechCrunch. If you choose a hamburger menu, then you need to keep informed about usage statistics. For more information on navigation usability essentials, check out “Navigation Essentials in Windows Store Apps” (msdn.microsoft.com/magazine/dn342878).

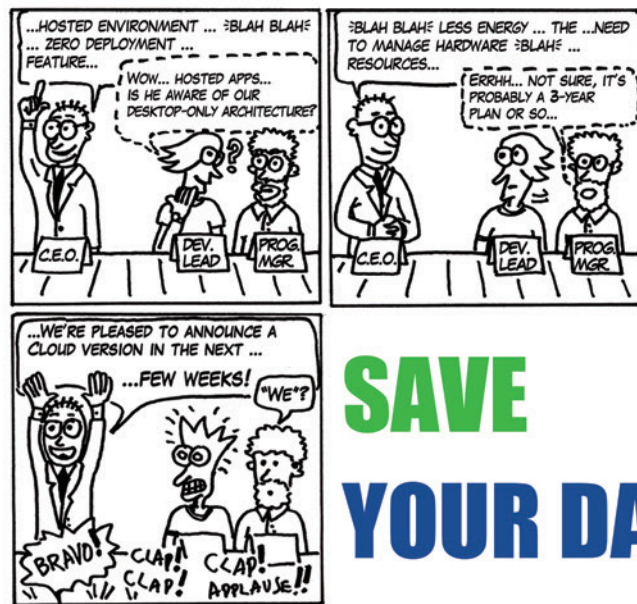
Wrapping Up

As a rule of thumb, more accessible software is more usable software. You can increase your user base by as much as 20 percent by implementing tiny changes. That’s a giant number by business standards, and any manager worth their salt (even pointy-haired bosses) would jump on those kinds of numbers. A great way to incorporate accessibility, and therefore better usability, is to try accessible technologies yourself. Blindfold yourself and try to use a page with a screen reader. Use a voice device.

While some practices here are listed as “good” or “bad,” that doesn’t mean they’re always one or the other. You can always break the mold of bad practices and make something more usable. It’s easy for good practices to go awry, as well. Ask your users what they like, but don’t always believe them because they can occasionally be wrong. Monitoring app usage is a great way to see what they’re really doing. Use your best judgment. ■

RACHEL APPEL is a consultant, author, mentor and former Microsoft employee with more than 20 years of experience in the IT industry. She speaks at top industry conferences such as Visual Studio Live!, DevConnections, MIX and more. Her expertise lies within developing solutions that align business and technology focusing on the Microsoft dev stack and open Web. For more about Appel, visit her Web site at rachelappel.com.

THANKS to the following Microsoft technical expert for reviewing this article:
Frank La Vigne



**SAVE
YOUR DAY!**



SUPPORT FOR
VISUAL STUDIO
2015

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-B618-66E8C16A410>

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



tools for developers, by developers

More information: www.softfluent.com Contact us: info@softfluent.com



Darwin's Camera

Every time I think that our technological world can't get any stupider, it surprises and delights me by doing exactly that. This latest increment of human dementia involves selfies.

By a selfie, of course, I mean a photo of oneself, taken with the front-facing camera on a smartphone, framed by looking in the phone's display. With a selfie, you never have to ask strangers to snap a photo of you in front of Brussels's "Manneken Pis" (bit.ly/1GgLSaA) or whatever, trusting them not to run off with your cool new iPhone.

The rise of the selfie has been spectacular. Exactly four years ago I bought my first smartphone, a then-state-of-the-art Motorola Droid X2. It didn't have a front-facing camera with which to take selfies, yet two years later, "selfie" was the word of the year for the Oxford English Dictionary. (Selfie, schmelfie. When are they going to include my coinages of hassle budget, marketingbozo, armadillo and MINFU? See my April Fools' Day column from 2013 at msdn.microsoft.com/magazine/dn166939.)

Once you start taking selfies, you quickly find that your arm isn't long enough to show much background. You then need a selfie stick, an extendable pole that holds the camera further out in front of you. Bloomberg named selfie sticks "the gift of the year for 2014" after 100,000 people bought them last December.

But as with any new infatuation, we don't notice its (her, his) warts until it's been around for a while. That's now happening with selfies.

Selfie sticks poke people and damage things, leading many establishments to ban them. Disney Parks, the Metropolitan Museum of Art, and the Wimbledon Tennis Tournament have already taken this step. Most soccer stadiums forbid them as potential riot weapons. In South Korea, selfie sticks have to be registered, making them harder to obtain than guns in some parts of the United States. ("Selfie sticks don't poke people's eyes out. People poke people's eyes out.")

Think this is crazy? I haven't even gotten started yet. The Russian government has noticed that its citizens are killing themselves taking ever-more-thrilling selfies. Now authorities are trying to convince selfie takers to be more careful by publishing a brochure, which you can view at the Russian Ministry of Internal Affairs' Web site (mvd.ru/safety_selfie). How well this engages the Russian psyche I will be curious to see.



Figure 1 Helpful iconography is great, but will determined selfie takers just look at it as motivation?

The brochure depicts all sorts of ways to kill yourself while taking a selfie, discouraged (one hopes) by the slashed-circle "No" symbol (see **Figure 1**). It suggests that we not take selfies while leaning out of car windows, standing in front of speeding trains, climbing electric power lines, pointing guns at our heads, standing next to hungry tigers and so on. I suspect it may be patterned after the Melbourne Metro's "Dumb Ways to Die" video (bit.ly/1h0sITK).

The paradox is obvious to us geeks. Anyone stupid enough to need a warning against taking a selfie in front of an oncoming train is probably too stupid to benefit from that warning. "Doh! [head slap] Silly me. I was just about to climb

that power pylon for my selfie, but your nice brochure showed me that I shouldn't. Thanks, Russian Ministry of Internal Affairs!" It reminds me of Clippy's old tip of the day in Microsoft Office 97, which would occasionally warn: "You can hurt yourself if you run with scissors." How many lives did that actually save?

The anti-selfie campaign could actually end up raising the IQ of Earth's population. Think about it. The stupidest people will see one of these brochures and say, "Bozhemoi, cool selfie ideas. I try one with train," and kill themselves off. Selfies as accidental eugenics, do you suppose?

I'll pour oil on this troubled fire, by coining new words to describe the trend of life-threatening self-snapshots. How about "fatelfie" (fatal + selfie) to describe a selfie that results in the death of the photographer. Maybe shorten it to "felfie?" To extend the idea (as always) toward absurdity, how about "Darwinelfie," a fatelfie so stupid it comes with an automatic Darwin award (darwinawards.com)? For example: "Wow, look at the tiger's claw puncturing the guy's neck just as he clicks the shutter. That's a definite Darwinelfie."

Oxford English Dictionary, take note. I'll be expecting the 2015 word of the year award. ■

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.

Of drivers who own family cars,

86%

would rather be *driving one of these.*



Move into the Fast Lane with MobileTogether®

In a little over two days, with one developer, we created a full-featured survey application and asked Android™, iOS®, and Windows® mobile users about their dream cars. 86% wanted to go faster.



ALTOVA®

MobileTogether®

Ditch the Minivan of Mobile Development

It's no longer practical to wait weeks or months for a cutting-edge mobile business solution. Your team needs to create, test, and deploy to all devices at the speed of business. With MobileTogether, we've removed the roadblocks.

- Drag and drop UI design process
- Powerful visual & functional programming paradigm
- Native apps for all major mobile platforms
- Connectivity to SQL databases, XML, HTML, and more
- Deploy full-featured business app solutions in record time
- Pricing so affordable you might just have money left to start saving for that sports car



www.altova.com/MobileTogether

FREE COMMUNITY LICENSE

A \$9,975 VALUE FOR FREE | SUPPORT AND UPDATES INCLUDED

CLAIM YOUR LICENSE

syncfusion.com/MSDNcommunity

Comprehensive offering includes more than 650 components across 12 platforms,
an easy-to-use big data platform, and much more.



WEB

ASP.NET

ASP.NET MVC

JavaScript

Silverlight



MOBILE

Orubase

WinRT

Windows Phone

Android

Xamarin

iOS



DESKTOP

WPF

Windows Forms



FILE FORMATS

Excel

Word

PDF

Powerpoint



DATA SCIENCE

Big Data Platform

Predictive Analytics