THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2016 VOL 31 NO 5

# msdn magazine




# When Only the Best Will Do

Our high-performance and feature-complete UI Controls and Libraries will help you build your best, without limits or compromise


DevExpress®

Free 30-day Trial
devexpress.com/try

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

# Unleash the UI Superhero in You

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

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

DevExpress® WIN ASP MVC WPF

THE MICROSOFT JOURNAL FOR DEVELOPERS

MAY 2016 VOL 31 NO 5

# msdn magazine

Dependency Injection in ASP.NET Core.....14



## COLUMNS



Microsoft

TEXT CONTROL

# ASP.NET MVC REPORTING

## The first cross-browser, true WYSIWYG HTML5-based document editor.

## Now with full ASP.NET MVC support.

Give your users an MS Word compatible editor to create powerful documents and reporting templates anywhere - in any browser. Feature-complete including mail merge, sections, headers and footers, drawings and shapes, tables, barcodes and charts.

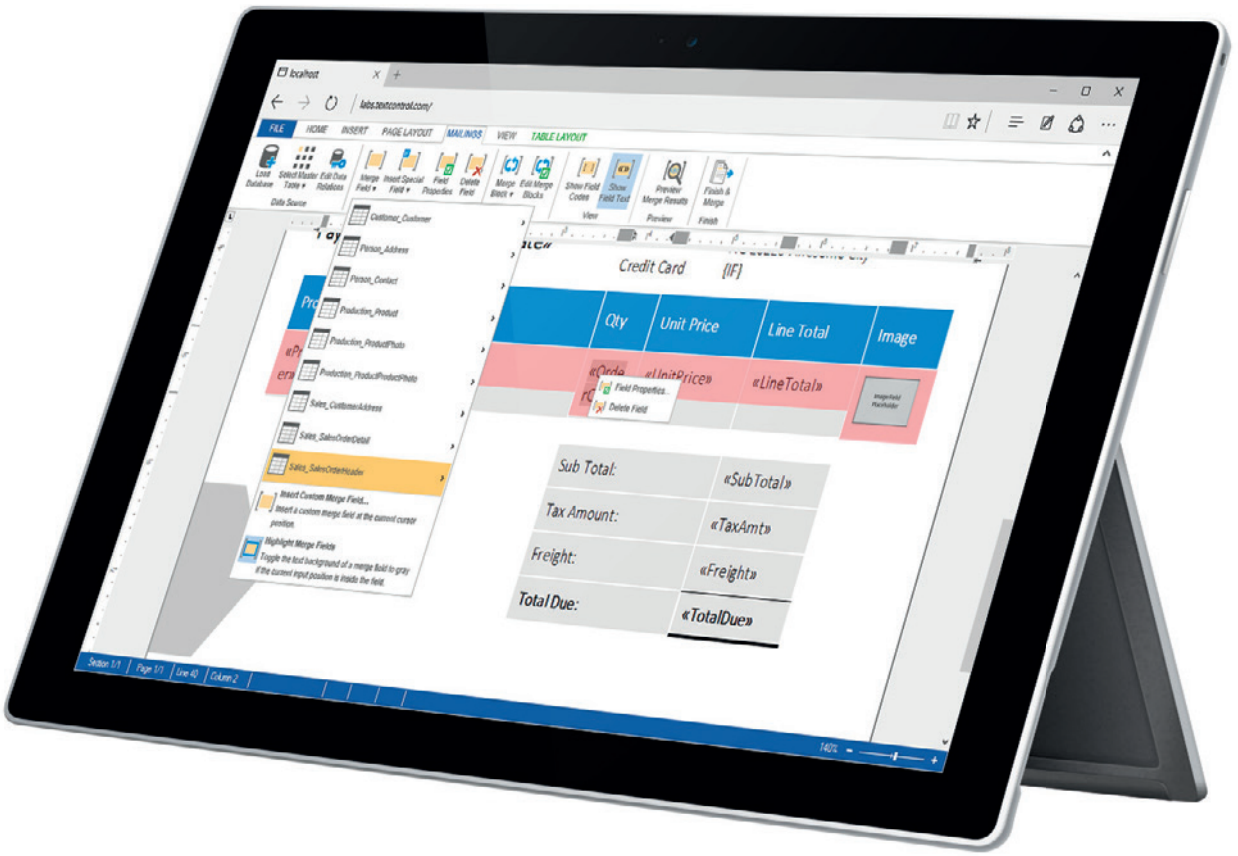
Available for ASP.NET Web Forms and MVC.

Visual Studio Partner

HTML 5

Call us: 855-533-TEXT

© 2016 Text Control GmbH. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective owners.




Edit and create MS Word documents

Create and modify Adobe® PDF documents

Create reports and mail merge templates

Integrate with Microsoft® Visual Studio

```
PM> Install-Package TXTextControl.Web
```

**Live demo** and 30-day trial version download at:

www.textcontrol.com/html5

X13 released

Software ▪ Training ▪ Consulting

TEXT CONTROL

msdn magazine

MAY 2016 VOLUME 31 NUMBER 5

**General Manager** Jeff Sandquist
**Director** Dan Fernandez
**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** Sharon Terdeman
**Features Editor** Ed Zintel
**Group Managing Editor** Wendy Hernandez
**Senior Contributing Editor** Dr. James McCaffrey
**Contributing Editors** Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt, Bruno Terkaly
**Vice President, Art and Brand Design** Scott Shultz
**Art Director** Joshua Gould

ENTERPRISE COMPUTING GROUP

**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**
*Print Production Coordinator* **Lee Alexander**

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

1105 MEDIA INC
YOUR GROWTH. OUR BUSINESS.

**Chief Executive Officer**
Rajeev Kapur

**Chief Operating Officer**
Henry Allain

**Vice President & Chief Financial Officer**
Michael Rafter

**Chief Technology Officer**
Erik A. Lindgren

**Executive Vice President**
Michael J. Valenti

**Chairman of the Board**
Jeffrey S. Klein

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

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

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

**CORPORATE ADDRESS** 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 www.1105media.com

**MEDIA KITS** Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

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

**LIST RENTAL** This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jlong@meritdirect.com; Web: www.meritdirect.com/1105

**Reaching the Staff**

Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at Redmondmag.com.

E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com

Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618

Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311

The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

BPA WORLDWIDE

1906 2006 100 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

# LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

## THE PREMIER **BARCODE** SDK

LEADTOOLS Barcode SDK is the world's premier toolkit for developing applications that require reading and writing **1D** and **2D barcodes**. With comprehensive support of over 100 different barcode types and sub-types such as UPC, EAN, Code 128, QR Code, Data Matrix, PDF417, Aztec, Maxicode, and Micro QR, LEADTOOLS is the right choice.

.NET Windows API Java WinRT Linux iOS OS X Android JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
**WWW.LEADTOOLS.COM**

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
**800.637.1840**



MICHAEL DESMOND

# Going Solid

The nuclear meltdown at Three Mile Island (TMI) in 1979 was a seminal event that's informed disciplines from crisis management and crew training to man-machine interface design. One fascinating aspect of the TMI disaster—and a very similar 1977 incident at the Davis-Besse plant in Ohio that foreshadowed it—was how the plant crew's response was guided by wisdom received from another domain.

> Hidden among the hard lessons of TMI is the risk that occurs when received wisdom is applied across domains.

A lot went wrong after the turbines tripped and shut down at TMI, but crucial to the event was a stuck valve that, unknown to the crew, was releasing coolant out the top of the reactor's pressurizer tank. That tank helps control the pressure of water in the reactor cooling system, ensuring it stays liquid even at extreme temperatures. It also maintains a bubble of compressible steam to absorb damaging forces—known as a water hammer—that can occur when fluid flow is disrupted, such as when a valve closes or a pump turns on. The TMI plant operators, many of whom were trained and served as reactor operators in the U.S. Navy, were drilled from day one to never let the pressurizer fill completely with water—a condition known as "going solid."

James Mahaffey, author of the book "Atomic Accidents" (Pegasus Books, 2015), says the naval proscription against going solid reflects the unique character and operating environment of submarine reactors and cooling systems, which are compact, lightweight and (compared to massive commercial reactors) very low power.

"Characteristics of a small reactor do not scale up perfectly to a big reactor," he says. "The primary coolant system on a power reactor is made of very big, thick pipes, and you try to reduce cost by minimizing the number of turns. The turbine hall is huge."

By contrast, the lightweight plumbing and tight bends in a submarine reactor cooling system pose a vulnerability. Says Mahaffey: "A primary coolant loop water hammer in a sub would possibly mean loss of the boat and everyone in it."

It's deadly serious stuff, and it colored the choices the Navy-trained crews at TMI and Davis-Besse were forced to make. Both reactor vessels reported low pressure and rising temperatures, indicators of dangerous coolant loss, yet the pressurizer tanks reported high water levels that would mean the reactor vessels were full of water. Unaware of the stuck-open pressurizer relief valve and a plant design flaw that trapped steam in the reactor vessels, each crew had to choose—cool the reactor or prevent the system from going solid. In each instance, plant operators chose the latter, overriding the automatic emergency core cooling system (ECCS) and denying critically needed water to the reactor core.

Said Mahaffey of the TMI incident: "They were under strict guidelines to *never* let the pressurizer go solid, and yet it was. The internal stress to meet this guideline was so severe, they left the rails and violated another guideline (shutting down the ECCS)."

Hidden among the hard lessons of TMI is the risk that occurs when received wisdom is applied across domains. The reactor operators were confronted with this risk in the most stark possible terms, but the challenge exists in every discipline. Developers moving among platforms, languages, organizations and projects must be aware of the preconceptions and habits they bring with them, and be careful not to fall into traps disguised as good practice.

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

© 2016 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.

# Switch to Amyuni PDF

AMYUNI


## Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

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

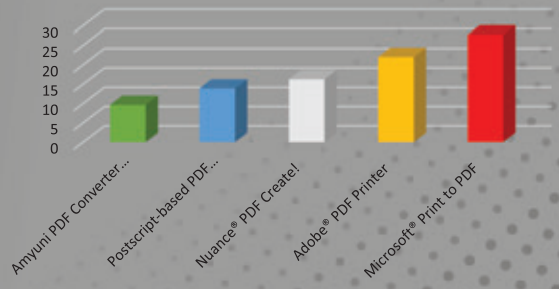
## Other Developer Components from Amyuni®

- 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


## 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




CERTIFIED FOR Windows Server 2008

Windows Server 2012 Certified

Windows 7

Windows

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

All development tools available at

www.amyuni.com



DINO ESPOSITO

# Building an Historical CRUD

Relational databases have been around since the 1970s and a few generations of developers started and ended their careers without learning, or just mildly considering, an alternate approach to data storage. Recently, large social networks provided strong evidence that relational databases couldn't serve all possible business scenarios. When a (really) huge amount of schemaless data comes your way, relational databases might sometimes be a bottleneck rather than a pipe.

Can you imagine what it might take to instantaneously count the likes on a comment friends made to a post in an all-encompassing relational database with a few billion records total? Not to mention that restricting the definition of a post to a rigid schema is at least challenging. For a mere matter of business survival, social networks at some point evolved and moved their storage focus to a mix of relational and non-relational data stores, thus officially opening the business of polyglot data.

The fundamental lesson learned from the software architecture of social networks is that sometimes plain storage of the data you have is not, business-wise, the ideal approach. Instead of just storing any data you've got, it's preferable to store details about an event occuring and the data involved with that particular event.

In this article, I'll first dig out the business foundation of event sourcing—using logged events as the primary data source of applications—and then discuss how to refresh existing Create, Read, Update, Delete (CRUD) skills in light of events. To make it clear from the very beginning, the question is not whether you need event sourcing. The question is when you need it and how you code it.

## Toward Dynamic Data Models

Using polyglot data is a hot topic today: relational databases for structured data, NoSQL data stores for less-structured data, key-value dictionaries for preferences and logs, graph databases for relationships and correlations. Introducing different storage models running side-by-side is a step in the right direction, but it seems to me more like an effective remedy to a visible symptom than the cure to the real and underlying disease.

The relational model owes its decade-long effectiveness to a balanced set of benefits it provides in reading and writing data. A relational model is easy to query and update even though it shows limits under some (very) extreme conditions. Overall performance comprehensibly decreases on tables with a few million records and a few hundred columns. In addition, the schema of data is fixed and knowledge of the database structure is required to create ad hoc and fast queries. In other words, in the world you code in today, a comprehensive model, like the big up-front relational model, is a huge constraint that first ends up limiting your expressivity and, later, your programming power. In the end, a model is just a model and it's not what you observe directly in the real world. In the real world, you don't observe any model. Instead, you use a model to encapsulate some well-understood and repeatable behavior. Ultimately, in the real world, you observe events but expect to store event-related information in a constrained (relational) model. When this proves hard to do, look into alternative storage models that just release some schema and indexing constraints.

## An Event-Based Storage Model

For decades, it was helpful and powerful to save just the current state of entities. When you save a given state, you overwrite the existing state, thus losing any previous information. This behavior doesn't deserve praise or blame, per se. For years, it proved to be an effective approach and gained wide acceptance. Only the business domain and customers can really say whether losing any past state is acceptable. Facts say that for many years and for most businesses, it was acceptable. This trend is changing as more and more business applications require tracking the full history of business entities. What was called CRUD for years—plain Create, Read, Update, Delete operations—and modeled on top of plain relational tables is now evolving in what can be generically referred to as historical CRUD. Historical CRUD is simply a CRUD code base where the implementation manages to track down the entire list of changes.

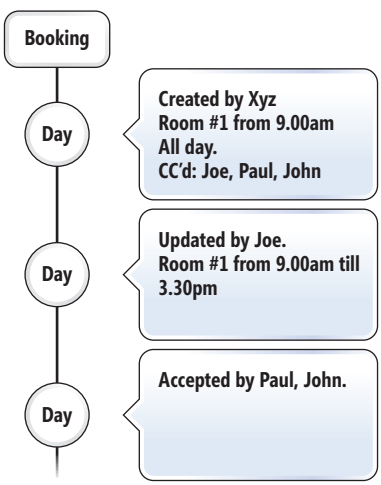



Figure 1 **A Timeline-Based View for the Entire History of a Booking**

The real world is full of line-of-business (LoB) systems that in some way track events as they take place in the domain. Such classes of application existed for decades—some even written in COBOL or Visual Basic 6. No doubt, for example, that an accounting application tracks all changes that might occur to invoices

DevExpress

# DevExpress Spreadsheet Control

## Excel® Inspired UI Controls for Desktops and the Web

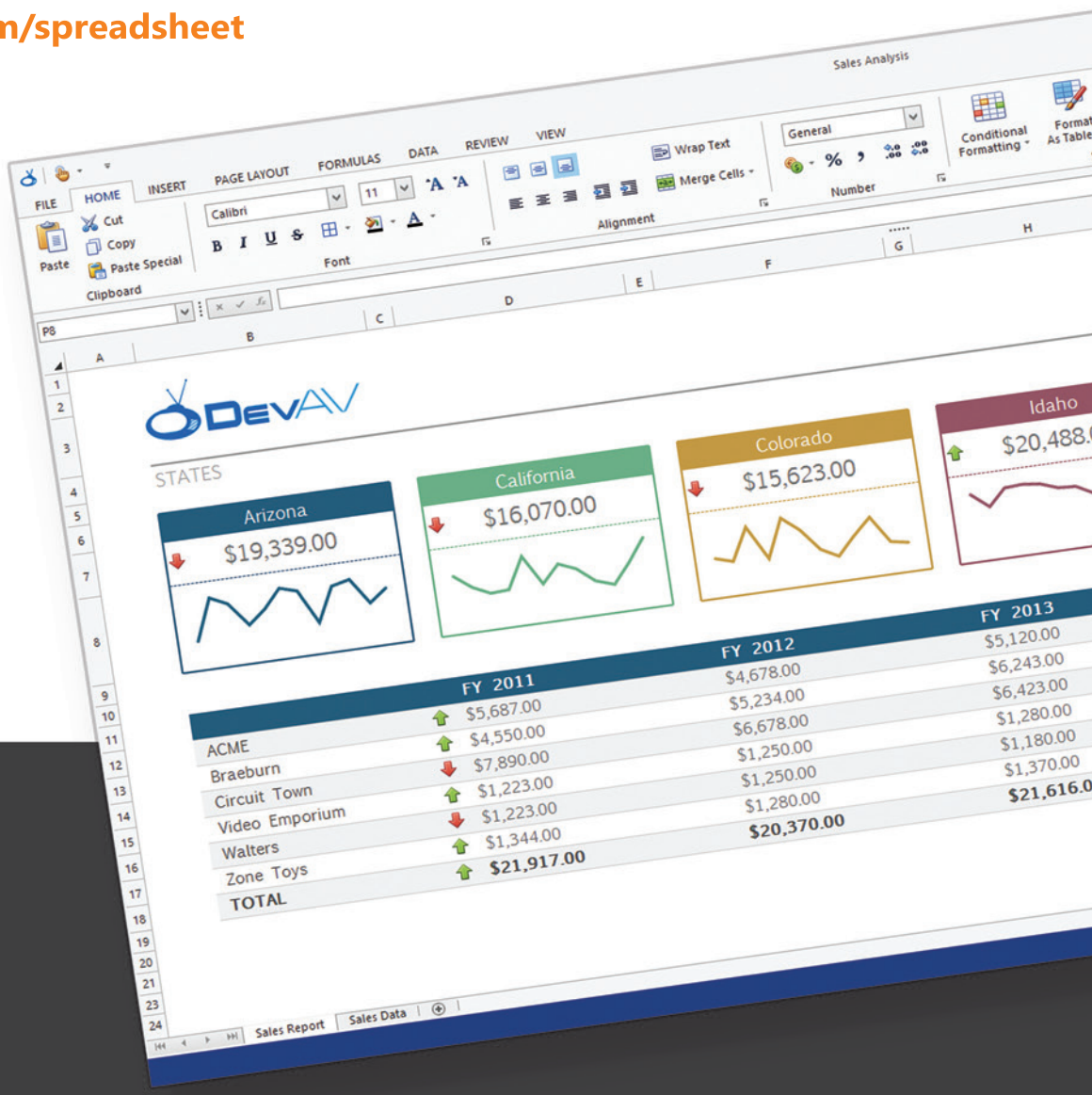
It's no secret...spreadsheets are an important part of business and if you need to incorporate the UX and functionality end-users have come to expect from Microsoft Excel®, DevExpress Spreadsheet is the tool for you. The component library ships with dozens of easy-to-implement features including chart support and fully integrates with other DevExpress components like our Office® inspired Ribbon.

WIN WPF ASP MVC

### Your Next Great Spreadsheet Starts @DevExpress

See how you can harness the power of spreadsheets in your next .NET app.

**devexpress.com/spreadsheet**




All trademarks or registered trademarks are propertyof their respective owners.

such as a change of date or address, the issuing of a credit note and the like. In some business scenarios, tracking events has been a requested feature since the early days of software, often falling under the broader umbrella of the auditing functionality.

Therefore, auditing business events isn't a new concept in software. For decades, development teams solved the same problem over and over, reworking and rehashing known techniques the best way they could possibly find. Today, the good old practice of auditing business events goes under the more engaging name of Event Sourcing.

## Coding Your Way to Business Events

So let's say you have a conceptually simple application like one that lets users book a shared resource, say a meeting room. When the user reviews the state of a given booking, she might be presented with not just the current state of the booking, but the entire list of updates since creation. **Figure 1** sketches a possible timeline-based UI for the view.

How would you design a booking data model that behaves as an historical CRUD rather than as a plain state-based CRUD? Adding more columns to the table definition isn't enough. The key difference between a CRUD and an historical CRUD is that in the latter scenario you want to store multiple copies of the same entity, one per each business event that it went through at a given time. **Figure 2** shows a possible new structure for the booking table of the relational database.

The table depicted in **Figure 2** has the expected set of columns that fully represent the state of a business entity, plus a few other ones. At the very minimum you want to have a primary key column to uniquely identify a row in the table. Next, you want to have a timestamp column that indicates either the time of the operation on the database or just any timestamp that makes sense for the business. More in general, the column serves the purpose of associating a safe date to the status of the entity. Finally, you want to have a column that describes the event that was logged.

It's still a relational table and it still manages the list of bookings the application needs. No new technology was added, but conceptually the table schematized in **Figure 2** is a quantum leap from a classic CRUD. Adding records to the new table is easy. You just fill out records and append them as you get notification that something has happened within the system that must be tracked down. So much for the C in CRUD; but what about other operations?

## Updates and Deletes in an Historical CRUD

Once the classic relational table has been turned into an historical, event-based table, role and relevance of updates and deletions change significantly. First off, updates disappear. Any updates to the logical state of the entity are now implemented as a new record appended that tracks the new state.

Deletions are a trickier point and the final word on how to code it resides in the business logic of the domain. In an ideal event-based world, there are no deletions. Data just adds up, therefore a deletion is simply the addition of a new event that informs you that the entity doesn't logically exist anymore. However, physical removal of data from a table isn't prohibited by law and can still happen. Note, though, that in an event-based scenario, the entity to remove is not made of a single record, but consists of a collection of records, as represented in **Figure 2**. If you decide to delete an entity, then you must remove all events (and records) that relate to it.

## Reading the State of an Entity

The biggest benefit you gain from logging business events in your application is that you never miss anything. You can potentially track the state of the system at any given time, figure out the exact sequence of actions that led to a given state and undo—in total or in part—those events. This lays the groundwork for self-made business intelligence and what-if scenarios in business analysis. To be more precise, you won't automatically get these features coming out of the box with your application, but you already have any data you need to develop such extensions on top of the existing application.

The hardest part of an historical CRUD is reading data. You're now tracking down all relevant business events in the sample booking system, but there's no place where you can easily get the full list of standing bookings. There's no quick-and-easy way to know, say, how many bookings you have next week. This is where projections fit in. **Figure 3** summarizes the overall architecture of a system that evolves from a plain CRUD to an historical CRUD.

An event-based system is inevitably geared toward implementing a neat separation between the command and query stack. From the presentation layer, the user triggers a task that proceeds through the application and domain layer involving all the business logic components on the way. A command is the trigger of a business task that alters the current state of the system, which means that something must be committed that logically modifies the existing state. As mentioned, in an event-based system—even when the system is a plain, simple CRUD system—altering the state means adding a record that indicates that the users created or updated a particular booking. The block in **Figure 3** labeled "Event Repository" represents any layer of code responsible for persisting the event. In terms of concrete technologies, the Event Repository block can be an Entity Framework-based repository class, as well as a wrapper around a document database (Azure DocumentDB, RavenDB or MongoDB). Even more interestingly, it can be a wrapper class using the API of an event store such as EventStore or NEventStore.

In an event-based architecture, the state of a given entity is algorithmically calculated upon request. This process goes under

| Event ID | Timestamp | What Happened | Entity ID | Set of columns representing the state of the entity |
|---|---|---|---|---|
| 123456 | Monday | Booking CREATED | 1 | Set of columns representing the *current* state of the entity |
| 234567 | Wednesday | Booking UPDATED | 1 | Set of columns representing the *current* state of the entity |
| 345678 | Thursday | Booking ACCEPTED | 1 | Set of columns representing the *current* state of the entity |

Figure 2 **A Possible Relational Data Model for a Historical CRUD Application**

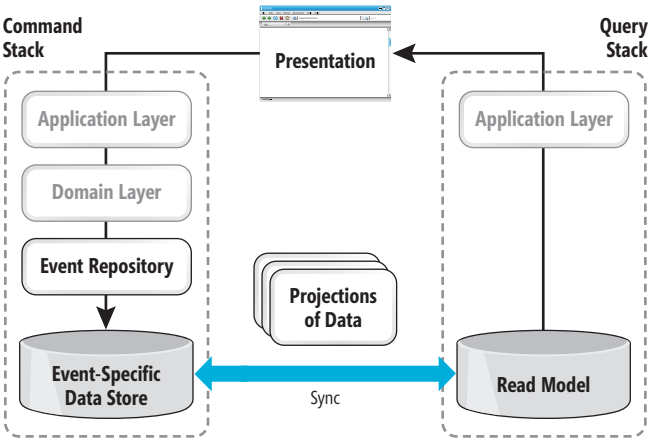



Figure 3 **Architecture of an Historical CRUD System**

the name of event replay and consists of querying all events that relate to the specified entity and applying all of them to a fresh new instance of the entity class. At the end of the loop, the entity instance is up-to-date because it has the state of a new instance that went through all the recorded events.

More in general, processing the log of events builds a projection of data and extracts a dynamic data model out of a lower-level amount of data. This is what is referred to as the Read Model in **Figure 3**. On top of the same log of events you can build all data models that serve the various front ends. To use a SQL analogy, building a projection of data from logged events is the same as building a view out of a relational table.

Replaying events to determine the current state of an entity for query purposes is generally a viable option but it becomes less and less effective as the number of events or the frequency of requests grows over time. You don't want to go through a few thousand records every time just to figure out the current balance of a bank account. Likewise, you don't want to walk through hundreds of events to present the list of pending bookings. To work around these issues, the read model often takes the form of a classic relational table that is programmatically kept in sync with the table of logged events.

## Wrapping Up

Most applications can still be grossly catalogued as CRUD apps. The same way Facebook can be presented in some way as a CRUD, perhaps just a little bit larger than average. Seriously, for most users the last known good state is still enough, but the number of customers for which this view is insufficient is growing. The next might just be your best customer. This article scratched just the surface of an historical CRUD. Next month, I'll present a concrete example. Stay tuned! ■

**Dino Esposito** *is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Modern Web Applications with ASP.NET" (Microsoft Press, 2016). A technical evangelist for the .NET 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: @despos.*

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



dtSearch®

Instantly Search Terabytes of Text

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

Highlights hits in all data types; 25+ search options

With APIs for .NET, Java 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



JULIE LERMAN

# Hybrid Dapper and Entity Framework Apps

You've probably noticed that I write a lot about Entity Framework, the Microsoft Object Relational Mapper (ORM) that's been the prime .NET data access API since 2008. There are other .NET ORMs out there but a particular category, micro-ORMs, gets a lot of notice for great performance. The micro-ORM I've heard mentioned most is Dapper. What finally piqued my interest enough to take some time out to crack it open recently was various developers reporting that they've created hybrid solutions with EF and Dapper, letting each ORM do what it's best at within a single application.

After reading numerous articles and blog posts, chatting with developers and playing a bit with Dapper, I wanted to share my discoveries with you, especially with those who, like me, perhaps heard of Dapper, but don't really know what it is or how it works—or why people love it. Keep in mind that I am in no way an expert. Rather, I know enough to satisfy my curiosity for the time being, and hopefully spark your interest so you'll dig even further.

## Why Dapper?

Dapper has an interesting history, having spawned from a resource you might be extremely familiar with: Marc Gravell and Sam Saffron built Dapper while working at Stack Overflow, solving performance issues with the platform. Stack Overflow is a seriously high-traffic site that's destined to have performance concerns. According to the Stack Exchange About page, Stack Overflow had 5.7 billion page views in 2015. In 2011, Saffron wrote a blog post about the work he and Gravell had done, titled, "How I Learned to Stop Worrying and Write My Own ORM" (bit.ly/1SftzIm), which explains the performance issues Stack was having at the time, stemming from its use of LINQ to SQL. He then details why writing a custom ORM, Dapper, was the answer for optimizing data access on Stack Overflow. Five years later, Dapper is now widely used and open source. Gravell and Stack and team member Nick Craver continue to actively manage the project at github.com/StackExchange/dapper-dot-net.

## Dapper in a Nutshell

Dapper focuses on letting you exercise your SQL skills to construct queries and commands as you think they should be. It's closer to "the metal" than a standard ORM, relieving the effort of interpreting queries such as LINQ to EF into SQL. Dapper does have some cool transformational features such as the ability to explode a list passed to a WHERE IN clause. But for the most part, the SQL you send to Dapper is ready to go and the queries get to the database much more quickly. If you're good at SQL, you can be sure you're writing the most performant commands possible. You need to create some type of IDbConnection, such as a SqlConnection with a known connection string, in order to execute the queries. Then, with its API, Dapper can execute the queries for you and—provided the schema of the query results can be matched up with the properties of the targeted type—automatically instantiate and populate objects with the query results. There is another notable performance benefit here: Dapper effectively caches the mapping it learned, resulting in very fast deserialization of subsequent queries. The class I'll populate, DapperDesigner (shown in **Figure 1**), is defined to manage designers who make very dapper clothing.

Code download available at msdn.com/magazine/0516magcode.

The project where I'm executing queries has a reference to Dapper, which I retrieve via NuGet (install-package dapper). Here's a sample call from Dapper to execute a query for all the rows in the DapperDesigners table:

```
var designers = sqlConn.Query<DapperDesigner>("select * from DapperDesigners");
```

Note that for code listings in this article, I'm using select * rather than explicitly projecting columns for queries when I want all columns from a table. sqlConn is an existing SqlConnection object that I've already instantiated, along with its connection string, but haven't opened.

The Query method is an extension method provided by Dapper. When this line executes, Dapper opens the connection, creates a DbCommand, executes the query exactly as I wrote it, instantiates a DapperDesigner object for each row in the results and pushes the values from the query results into the properties of the objects. Dapper can match the result values with properties by means of a few patterns even if the property names don't match the column names and even if the properties aren't in the same order as the matching columns. It can't read minds, though, so don't expect it to figure out mappings involving, for example, numerous string values where the orders or names of the columns and properties are out of sync. I did try a few odd experiments with it to see how it would respond and there are also global settings that control how Dapper can infer mappings.

## Dapper and Relational Queries

Figure 1 **DapperDesigner Class**

```
public class DapperDesigner
{
  public DapperDesigner() {
    Products = new List<Product>();
    Clients = new List<Client>();
  }
  public int Id { get; set; }
  public string LabelName { get; set; }
  public string Founder { get; set; }
  public Dapperness Dapperness { get; set; }
  public List<Client> Clients { get; set; }
  public List<Product> Products { get; set; }

  public ContactInfo ContactInfo { get; set; }
}
```

My DapperDesigner type has a number of relationships. There's a one-to-many (with Products), a one-to-one (ContactInfo) and many-to-many (Clients). I've experimented with executing queries across those relationships, and Dapper can handle the relationships. It's definitely not as easy as expressing a LINQ to EF query with an Include method or even a projection. My TSQL skills were pushed to the limit, however, because EF has allowed me to become so lazy over the past years.

Here's an example of querying across the one-to-many relationship using the SQL I would use right in the database:

```
var sql = @"select * from DapperDesigners D
          JOIN Products P
          ON P.DapperDesignerId = D.Id";

var designers= conn.Query<DapperDesigner, Product,DapperDesigner>
(sql,(designer, product) => { designer.Products.Add(product);
                              return designer; });
```

Notice that the Query method requires I specify both types that must be constructed, as well as indicate the type to be returned—expressed by the final type parameter (DapperDesigner). I use a multi-line lambda to first construct the graphs, adding the relevant products to their parent designer objects, and then to return each designer to the IEnumerable the Query method returns.

The downside of doing this with my best attempt at the SQL is that the results are flattened, just like they'd be with the EF Include method. I'll get one row per product with designers duplicated. Dapper has a MultiQuery method that can return multiple result-sets. Combined with Dapper's GridReader, the performance of these queries will definitely outshine EF Includes.

## Harder to Code, Faster to Execute

Expressing SQL and populating related objects are tasks I've let EF handle in the background, so this is definitely more effort to code. But if you're working with lots of data and runtime performance is important, it can certainly be worth that effort. I have about 30,000 designers in my sample database. Only a few of them have products. I did some simple benchmark tests where I ensured I was comparing apples to apples. Before looking at the test results, there are some important points to understand about how I did these measurements.

Remember that, by default, EF is designed to track objects that are the results of queries. This means it creates additional tracking objects, which involves some effort, and it also needs to interact with those tracking objects. Dapper, in contrast, just dumps results into memory. So it's important to take EF's change tracking out of the loop when making any performance comparisons. I do this by defining all my EF queries with the AsNoTracking method. Also, when comparing performance, you need to apply a number of standard benchmark patterns, such as warming up the database, repeating the query many times and throwing out the slowest and fastest times. You can see the details of how I built my benchmarking tests in the sample download. Still, I consider these to be "lightweight" benchmark tests, just to provide an idea of the differences. For serious benchmarks, you'd need to iterate many more times than my 25 (start at 500), and to factor in the performance of the system on which you're running. I'm running these tests on a laptop using a SQL Server LocalDB instance, so my results are useful only for comparison's sake.

The times I'm tracking in my tests are for executing the query and building the results. Instantiating connections or DbContexts isn't counted. The DbContext is reused so the time for EF to build the in-memory model is also not taken into account, as this would only happen once per application instance, not for every query.

**Figure 2** shows the "select *" tests for Dapper and the EF LINQ query so you can see the basic construct of my testing pattern. Notice that outside of the actual time gathering, I'm collecting the time for each iteration into a list (called "times") for further analysis.

There's one other point to be made about comparing "apples to apples." Dapper takes in raw SQL. By default, EF queries are expressed with LINQ to EF and must go through some effort to build the SQL for you. Once that SQL is built, even SQL that relies on parameters, it's cached in the application's memory so that effort is reduced on repetition. Additionally, EF has the ability to execute queries using raw SQL, so I've taken both approaches into account. **Figure 3** lists the comparative results of four sets of tests. The download contains even more tests.

In the scenarios shown in **Figure 3**, it's easy to make a case for using Dapper over LINQ to Entities. But the narrow differences between raw SQL queries may not always justify switching to Dapper for particular tasks in a system where you're otherwise using EF. Naturally, your own needs will be different and could impact the degree of variation between EF queries and Dapper. However, in a high-traffic system such as Stack Overflow, even the handful of milliseconds saved per query can be critical.

## Dapper and EF for Other Persistence Needs

So far I've measured simple queries where I just pulled back all columns from a table that exactly matched properties of the types being returned. What about if you're projecting queries into types? As long as the schema of the results matches the type, Dapper sees no difference in creating the objects. EF, however, has to work

Figure 2 **Tests to Compare EF and Dapper When Querying All DapperDesigners**

```
[TestMethod,TestCategory("EF"),TestCategory("EF,NoTrack")]
public void GetAllDesignersAsNoTracking() {
  List<long> times = new List<long>();
  for (int i = 0; i < 25; i++) {
    using (var context = new DapperDesignerContext()) {
      _sw.Reset();
      _sw.Start();
      var designers = context.Designers.AsNoTracking().ToList();
      _sw.Stop();
      times.Add(_sw.ElapsedMilliseconds);
      _trackedObjects = context.ChangeTracker.Entries().Count();
    }
  }
  var analyzer = new TimeAnalyzer(times);
  Assert.IsTrue(true);
}

[TestMethod,TestCategory("Dapper")
public void GetAllDesigners() {
  List<long> times = new List<long>();
  for (int i = 0; i < 25; i++) {
    using (var conn = Utils.CreateOpenConnection()) {
      _sw.Reset();
      _sw.Start();
      var designers = conn.Query<DapperDesigner>("select * from DapperDesigners");
      _sw.Stop();
      times.Add(_sw.ElapsedMilliseconds);
      _retrievedObjects = designers.Count();
    }
  }
  var analyzer = new TimeAnalyzer(times);
  Assert.IsTrue(true);
}
```

Figure 3 **Average Time in Milliseconds to Execute a Query and Populate an Object Based on 25 Iterations, Eliminating the Fastest and Slowest**

| *AsNoTracking queries | Relationship | LINQ to EF* | EF Raw SQL* | Dapper Raw SQL |
|---|---|---|---|---|
| All Designers (30K rows) | – | 96 | 98 | 77 |
| All Designers with Products (30K rows) | 1 : * | 251 | 107 | 91 |
| All Designers with Clients (30K rows) | * : * | 255 | 106 | 63 |
| All Designers with Contact (30K rows ) | 1 : 1 | 322 | 122 | 116 |

harder if the results of the projection don't align with a type that's part of the model.

The DapperDesignerContext has a DbSet for the DapperDesigner type. I have another type in my system called MiniDesigner that has a subset of DapperDesigner properties:

```
public class MiniDesigner {
    public int Id { get; set; }
    public string Name { get; set; }
    public string FoundedBy { get; set; }
  }
```

MiniDesigner isn't part of my EF data model, so DapperDesignerContext has no knowledge of this type. I found that querying all 30,000 rows and projecting into 30,000 MiniDesigner objects was 25 percent faster with Dapper than with EF using raw SQL. Again, I recommend doing your own performance profiling to make decisions for your own system.

Dapper can also be used to push data into the database with methods that allow you to identify which properties must be used for the parameters specified by the command, whether you're using a raw INSERT or UPDATE command or you're executing a function or stored procedure on the database. I did not do any performance comparisons for these tasks.

## Hybrid Dapper Plus EF in the Real World

There are lots and lots of systems that use Dapper for 100 percent of their data persistence. But recall that my interest was piqued because of developers talking about hybrid solutions. In some cases, these are systems that have EF in place and are looking to tweak particular problem areas. In others, teams have opted to use Dapper for all queries and EF for all saves.

In response to my asking about this on Twitter, I received varied feedback.

@garypochron told me his team was "moving ahead with using Dapper in high call areas & using resource files to maintain org of SQL." I was surprised to learn that Simon Hughes (@s1monhughes), author of the popular EF Reverse POCO Generator, goes the opposite direction—defaulting to Dapper and using EF for tricky problems. He told me "I use Dapper where possible. If it's a complex update I use EF."

I've also seen a variety of discussions where the hybrid approach is driven by separation of concerns rather than enhancing performance. The most common of these leverage the default reliance of ASP.NET Identity on EF and then use Dapper for the rest of the persistence in the solution.

Working more directly with the database has other advantages besides performance. Rob Sullivan (@datachomp) and Mike Campbell (@angrypets), both SQL Server experts, love Dapper. Rob points out that you can take advantage of database features that EF doesn't give access to, such as full text search. In the long run, that particular feature is, indeed, about performance.

On the other hand, there are things you can do with EF that you can't do with Dapper besides the change tracking. A good example is one I took advantage of when building the solution I created for this article—the ability to migrate your database as the model changes using EF Code First Migrations.

Dapper isn't for everyone, though. @damiangray told me Dapper isn't an option for his solution because he needs to be able to return IQueryables from one part of his system to another, not actual data. This topic, deferred query execution, has been brought up in Dapper's GitHub repository at bit.ly/22CJzJl, if you want to read more about it. When designing a hybrid system, using some flavor of Command Query Separation (CQS) where you design separate models for particular types of transactions (something I'm a fan of) is a good path. That way, you aren't trying to build data access code that's vanilla enough to work with both EF and Dapper, which often results in sacrificing benefits of each ORM. Just as I was working on this article, Kurt Dowswell published a post called "Dapper, EF and CQS" (bit.ly/1LEjYvA). Handy for me, and handy for you.

For those looking ahead to CoreCLR and ASP.NET Core, Dapper has evolved to include support for these, as well. You can find more information in a thread in Dapper's GitHub repository at bit.ly/1T5m5Ko.

## So, Finally, I Looked at Dapper. What Do I Think?

And what about me? I regret not taking the time to look at Dapper earlier and am happy I've finally done so. I've always recommended AsNoTracking or using views or procedures in the database to alleviate performance problems. This has never failed me or my clients. But now I know I have another trick up my sleeve to recommend to developers who are interested in squeezing even more performance out of their systems that use EF. It's not a shoo-in, as we say. My recommendation will be to explore Dapper, measure the performance difference (at scale) and find a balance between performance and ease of coding. Consider StackOverflow's obvious use: querying questions, comments and answers, then returning graphs of one question with its comments and answers along with some metadata (edits) and user info. They're doing the same types of queries and mapping out the same shape of results over and over again. Dapper is designed to shine with this type of repetitive querying, getting smarter and faster each time. Even if you don't have a system with the insane number of transactions that Dapper was designed for, you're likely to find that a hybrid solution gives you just what you need. ■

**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," as well as a Code First and a DbContext edition, all from O'Reilly Media. Follow her on Twitter: @julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

**Thanks** to the following Stack Overflow technical experts for reviewing this article: *Nick Craver and Marc Gravell*

# We Made WPF GREAT!

Xceed
Business Suite
for WPF

Match the vision you have for your WPF application's user experience, and surpass corporate expectations.

Xceed

# Writing Clean Code in ASP.NET Core with Dependency Injection

Steve Smith

ASP.NET Core 1.0 is a complete rewrite of ASP.NET, and one of the main goals for this new framework is a more modular design. That is, apps should be able to leverage only those parts of the framework they need, with the framework providing dependencies as they're requested. Further, developers building apps using ASP.NET Core should be able to leverage this same functionality to keep their apps loosely coupled and modular. With ASP.NET MVC, the ASP.NET team greatly improved the framework's support for writing loosely coupled code, but it was still very easy to fall into the trap of tight coupling, especially in controller classes.

This article is based on ASP.NET Core 1.0, RC1. Some information may change when RC2 becomes available.

**This article discusses:**

- Examples of tight coupling
- Keeping classes honest
- ASP.NET Core dependency injection
- Unit-testable code
- Testing controller responsibilities

**Technologies discussed:**

ASP.NET Core 1.0

**Code download available at:**

msdn.com/magazine/0516magcode

## Tight Coupling

Tight coupling is fine for demo software. If you look at the typical sample application showing how to build ASP.NET MVC (versions 3 through 5) sites, you'll most likely find code like this (from the NerdDinner MVC 4 sample's DinnersController class):

```
private NerdDinnerContext db = new NerdDinnerContext();
private const int PageSize = 25;

public ActionResult Index(int? page)
{
  int pageIndex = page ?? 1;

  var dinners = db.Dinners
    .Where(d => d.EventDate >= DateTime.Now).OrderBy(d => d.EventDate);
  return View(dinners.ToPagedList(pageIndex, PageSize));
}
```

> When looking at code to evaluate its coupling, remember the phrase "new is glue."

This kind of code is very difficult to unit test, because the NerdDinnerContext is created as part of the class's construction, and requires a database to which to connect. Not surprisingly, such demo applications don't often include any unit tests. However, your application might benefit from a few unit tests, even if you're not test-driving your development, so it would be better to write

the code so it could be tested. What's more, this code violates the Don't Repeat Yourself (DRY) principle, because *every* controller class that performs any data access has the same code in it to create an Entity Framework (EF) database context. This makes future changes more expensive and error-prone, especially as the application grows over time.

When looking at code to evaluate its coupling, remember the phrase "new is glue." That is, anywhere you see the "new" keyword instantiating a class, realize you're *gluing* your implementation to that specific implementation code. The Dependency Inversion Principle (bit.ly/DI-Principle) states: "Abstractions should not depend on details; details should depend on abstractions." In this example, the details of how the controller pulls together the data to pass to the view depend on the details of how to get that data—namely, EF.

In addition to the new keyword, "static cling" is another source of tight coupling that makes applications more difficult to test and maintain. In the preceding example, there's a dependency on the executing machine's system clock, in the form of a call to DateTime.Now. This coupling would make creating a set of test Dinners to use in some unit tests difficult, because their EventDate properties would need to be set relative to the current clock's setting. This coupling could be removed from this method in several ways, the simplest of which is to let whatever new abstraction returns the Dinners worry about it, so it's no longer a part of this method. Alternatively, I could make the value a parameter, so the method might return all Dinners after a provided DateTime parameter, rather than always using DateTime.Now. Last, I could create an abstraction for the current time, and reference the current time through that abstraction. This can be a good approach if the application references DateTime.Now frequently. (It's also worth noting that because these dinners presumably happen in various time zones, the DateTimeOffset type might be a better choice in a real application).

## Be Honest

Another problem with the maintainability of code like this is that it isn't honest with its collaborators. You should avoid writing classes that can be instantiated in invalid states, as these are frequent sources of errors. Thus, anything your class needs in order to perform its tasks should be supplied through its constructor. As the Explicit Dependencies Principle (bit.ly/ED-Principle) states, "Methods and classes should explicitly require any collaborating objects they need in order to function correctly." The DinnersController class has only a default constructor, which implies that it shouldn't need any collaborators in order to function properly. But what happens if you put that to the test? What will this code do, if you run it from a new console application that references the MVC project?

```
var controller = new DinnersController();
var result = controller.Index(1);
```

The first thing that fails in this case is the attempt to instantiate the EF context. The code throws an InvalidOperationException: "No connection string named 'NerdDinnerContext' could be found in the application config file." I've been deceived! This class needs more to function than what its constructor claims! If the class needs a way to access collections of Dinner instances, it should request that through its constructor (or, alternatively, as parameters on its methods).

## Dependency Injection

Dependency injection (DI) refers to the technique of passing in a class's or method's dependencies as parameters, rather than hard-coding these relationships via *new* or static calls. It's an increasingly common technique in .NET development, because of the decoupling it affords to applications that employ it. Early versions of ASP.NET didn't make use of DI, and although ASP.NET MVC and Web API made progress toward supporting it, neither went so far as to build full support, including a container for managing the dependencies and their object lifecycles, into the product. With ASP.NET Core 1.0, DI isn't just supported out of the box, it's used extensively by the product itself.

With ASP.NET Core 1.0, dependency injection isn't just supported out of the box, it's used extensively by the product itself.

ASP.NET Core not only supports DI, it also includes a DI container—also referred to as an Inversion of Control (IoC) container or a services container. Every ASP.NET Core app configures its dependencies using this container in the Startup class's ConfigureServices method. This container provides the basic support required, but it can be replaced with a custom implementation if desired. What's more, EF Core also has built-in support for DI, so configuring it within an ASP.NET Core application is as simple as calling an extension method. I've created a spinoff of NerdDinner, called GeekDinner, for this article. EF Core is configured as shown here:

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddEntityFramework()
    .AddSqlServer()
    .AddDbContext<GeekDinnerDbContext>(options =>
      options.UseSqlServer(ConnectionString));

  services.AddMvc();
}
```

With this in place, it's quite simple to use DI to request an instance of GeekDinnerDbContext from a controller class like DinnersController:

```
public class DinnersController : Controller
{
  private readonly GeekDinnerDbContext _dbContext;

  public DinnersController(GeekDinnerDbContext dbContext)
  {
    _dbContext = dbContext;
  }

  public IActionResult Index()
  {
    return View(_dbContext.Dinners.ToList());
  }
}
```

Notice there's not a single instance of the new keyword; the dependencies the controller needs are all passed in via its constructor, and the ASP.NET DI container takes care of this for me. While I'm focused on writing the application, I don't need to worry

about the plumbing involved in fulfilling the dependencies my classes request through their constructors. Of course, if I want to, I can customize this behavior, even replacing the default container with another implementation entirely. Because my controller class now follows the explicit dependencies principle, I know that for it to function I need to provide it with an instance of a GeekDinnerDbContext. I can, with a bit of setup for the DbContext, instantiate the controller in isolation, as this Console application demonstrates:

```
var optionsBuilder = new DbContextOptionsBuilder();
optionsBuilder.UseSqlServer(Startup.ConnectionString);
var dbContext = new GeekDinnerDbContext(optionsBuilder.Options);
var controller = new DinnersController(dbContext);
var result = (ViewResult) controller.Index();
```

There's a little bit more work involved in constructing an EF Core DbContext than an EF6 one, which just took a connection string. This is because, just like ASP.NET Core, EF Core has been designed to be more modular. Normally, you won't need to deal with DbContextOptionsBuilder directly, because it's used behind the scenes when you configure EF through extension methods like AddEntityFramework and AddSqlServer.

## But Can You Test It?

Testing your application manually is important—you want to be able to run it and see that it actually runs and produces the expected output. But having to do that every time you make a change is a waste of time. One of the great benefits of loosely coupled applications is that they tend to be more amenable to unit testing than tightly coupled apps. Better still, ASP.NET Core and EF Core are both much easier to test than their predecessors were. To get started, I'll write a simple test directly against the controller by passing in a DbContext that has been configured to use an in-memory store. I'll configure the GeekDinnerDbContext using the DbContextOptions parameter it exposes via its constructor as part of my test's setup code:

```
var optionsBuilder = new DbContextOptionsBuilder<GeekDinnerDbContext>();
optionsBuilder.UseInMemoryDatabase();
_dbContext = new GeekDinnerDbContext(optionsBuilder.Options);

// Add sample data
_dbContext.Dinners.Add(new Dinner() { Title = "Title 1" });
_dbContext.Dinners.Add(new Dinner() { Title = "Title 2" });
_dbContext.Dinners.Add(new Dinner() { Title = "Title 3" });
_dbContext.SaveChanges();
```

With this configured in my test class, it's easy to write a test showing that the correct data is returned in the ViewResult's Model:

```
[Fact]
public void ReturnsDinnersInViewModel()
{
  var controller = new OriginalDinnersController(_dbContext);

  var result = controller.Index();

  var viewResult = Assert.IsType<ViewResult>(result);
  var viewModel = Assert.IsType<IEnumerable<Dinner>>(
    viewResult.ViewData.Model).ToList();
  Assert.Equal(1, viewModel.Count(d => d.Title == "Title 1"));
  Assert.Equal(3, viewModel.Count);
}
```

Of course, there's not a lot of logic to test here yet, so this test isn't really testing that much. Critics will argue that this isn't a terribly valuable test, and I would agree with them. However, it's a starting point for when there's more logic in place, as there soon will be. But first, although EF Core can support unit testing with its in-memory option, I'll still avoid direct coupling to EF in my controller. There's no reason to couple UI concerns with data access infrastructure concerns—in fact, it violates another principle, Separation of Concerns.

## Don't Depend on What You Don't Use

The Interface Segregation Principle (bit.ly/LS-Principle) states that classes should depend only on functionality they actually use. In the case of the new DI-enabled DinnersController, it's still depending on the entire DbContext. Instead of gluing the controller implementation to EF, an abstraction that provided the necessary functionality (and little or nothing more) could be used.

What does this action method really need in order to function? Certainly not the entire DbContext. It doesn't even need access to the full Dinners property of the context. All it needs is the ability to display the appropriate page's Dinner instances. The simplest .NET abstraction representing this is IEnumerable<Dinner>. So, I'll define an interface that simply returns an IEnumerable<Dinner>, and that will satisfy (most of) the requirements of the Index method:

```
public interface IDinnerRepository
{
  IEnumerable<Dinner> List();
}
```

I'm calling this a *repository* because it follows that pattern: It abstracts data access behind a collection-like interface. If for some reason you don't like the repository pattern or name, you can call it IGetDinners or IDinnerService or whatever name you prefer (my tech reviewer suggests ICanHasDinner). Regardless of how you name the type, it will serve the same purpose.

> One of the great benefits of loosely coupled applications is that they tend to be more amenable to unit testing than tightly coupled apps.

With this in place, I now adjust DinnersController to accept an IDinnerRepository as a constructor parameter, instead of a GeekDinnerDbContext, and call the List method instead of accessing the Dinners DbSet directly:

```
private readonly IDinnerRepository _dinnerRepository;
public DinnersController(IDinnerRepository dinnerRepository)
{
  _dinnerRepository = dinnerRepository;
}

public IActionResult Index()
{
  return View(_dinnerRepository.List());
}
```

At this point, you can build and run your Web application, but you'll encounter an exception if you navigate to /Dinners: InvalidOperationException: Unable to resolve service for type 'GeekDinner.Core.Interfaces.IdinnerRepository' while attempting to activate GeekDinner.Controllers.DinnersController. I haven't yet implemented the interface, and once I do, I'll also need to configure

/n software

# We didn't invent the Internet...

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

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

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

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

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

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity powered by

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

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

my implementation to be used when DI fulfills requests for IDinnerRepository. Implementing the interface is trivial:

```
public class DinnerRepository : IDinnerRepository
{
  private readonly GeekDinnerDbContext _dbContext;
  public DinnerRepository(GeekDinnerDbContext dbContext)
  {
    _dbContext = dbContext;
  }

  public IEnumerable<Dinner> List()
  {
    return _dbContext.Dinners;
  }
}
```

Note that it's perfectly fine to couple a repository implementation to EF directly. If I need to swap out EF, I'll just create a new implementation of this interface. This implementation class is a part of my application's infrastructure, which is the one place in the application where my classes depend on specific implementations.

To configure ASP.NET Core to inject the correct implementation when classes request an IDinnerRepository, I need to add the following line of code to the end of the ConfigureServices method shown earlier:

```
services.AddScoped<IDinnerRepository, DinnerRepository>();
```

This statement instructs the ASP.NET Core DI container to use a DinnerRepository instance whenever the container is resolving a type that depends on an IDinnerRepository instance. Scoped means that one instance will be used for each Web request ASP.NET handles. Services can also be added using Transient or Singleton lifetimes. In this case, Scoped is appropriate because my DinnerRepository depends on a DbContext, which also uses the Scoped lifetime. Here's a summary of the available object lifetimes:

- **Transient:** A new instance of the type is used every time the type is requested.
- **Scoped:** A new instance of the type is created the first time it's requested within a given HTTP request, and then re-used for all subsequent types resolved during that HTTP request.
- **Singleton:** A single instance of the type is created once, and used by all subsequent requests for that type.

The built-in container supports several ways of constructing the types it will provide. The most typical case is to simply provide the container with a type, and it will attempt to instantiate that type, providing any dependencies that type requires as it goes. You can also provide a lambda expression for constructing the type, or, for a Singleton lifetime, you can provide the fully constructed instance in ConfigureServices when you register it.

With dependency injection wired up, the application runs just as before. Now, as **Figure 1** shows, I can test it with this new abstraction in place, using a fake or mock implementation of the IDinnerRepository interface, instead of relying on EF directly in my test code.

This test works regardless of where the list of Dinner instances comes from. You could rewrite the data access code to use another database, Azure Table Storage, or XML files, and the controller would work the same. Of course, in this case it's not doing a whole lot, so you might be wondering …

## What About Real Logic?

So far I haven't really implemented any real business logic—it's just been simple methods returning simple collections of data. The real value of testing comes when you have logic and special cases you need to have confidence will behave as intended. To demonstrate this, I'm going to add some requirements to my GeekDinner site. The site will expose an API that will allow anybody to RSVP for a dinner. However, dinners will have an optional maximum capacity and RSVPs should not exceed this capacity. Users requesting RSVPs beyond the maximum capacity should be added to a waitlist. Finally, dinners can specify a deadline by which RSVPs must be received, relative to their start time, after which they stop accepting RSVPs.

I could code all of this logic into an action, but I believe that's way too much responsibility to put into one method, especially a UI method that should be focused on UI concerns, not business logic. The controller should verify that the inputs it receives are valid, and it should ensure the responses it returns are appropriate to the client. Decisions beyond that, and especially business logic, don't belong in controllers.

The best place to keep business logic is in the application's domain model, which shouldn't depend on infrastructure concerns (like databases or UIs). The Dinner class makes the most sense to manage the RSVP concerns described in the requirements, because it will store the maximum capacity for the dinner and it will know how many RSVPs have been made so far. However, part of the logic also depends on when the RSVP occurs—whether or not it's past the deadline—so the method also needs access to the current time.

I could just use DateTime.Now, but this would make the logic difficult to test and would couple my domain model to the system clock. Another option is to use an IDateTime abstraction and inject this into the Dinner entity. However, in my experience it's best to keep entities like Dinner free of dependencies, especially if you plan on using an O/RM tool like EF to pull them from a persistence layer. I don't want to have to populate dependencies of the entity as part of that process, and EF certainly won't be able to do it without additional code on my part. A common approach at this point is to pull the logic out of the Dinner entity and put it in some kind of service

Figure 1 **Testing DinnersController Using a Mock Object**

```
public class DinnersControllerIndex
{
  private List<Dinner> GetTestDinnerCollection()
  {
    return new List<Dinner>()
    {
      new Dinner() {Title = "Test Dinner 1" },
      new Dinner() {Title = "Test Dinner 2" },
    };
  }

  [Fact]
  public void ReturnsDinnersInViewModel()
  {
    var mockRepository = new Mock<IDinnerRepository>();
    mockRepository.Setup(r =>
      r.List()).Returns(GetTestDinnerCollection());
    var controller = new DinnersController(mockRepository.Object, null);

    var result = controller.Index();

    var viewResult = Assert.IsType<ViewResult>(result);
    var viewModel = Assert.IsType<IEnumerable<Dinner>>(
      viewResult.ViewData.Model).ToList();
    Assert.Equal("Test Dinner 1", viewModel.First().Title);
    Assert.Equal(2, viewModel.Count);
  }
}
```

(such as DinnerService or RsvpService) that can have dependencies injected easily. This tends to lead to the anemic domain model antipattern (bit.ly/anemic-model), though, in which entities have little or no behavior and are just bags of state. No, in this case the solution is straightforward—the method can simply take in the current time as a parameter and let the calling code pass this in.

With this approach, the logic for adding an RSVP is straightforward (see **Figure 2**). This method has a number of tests that demonstrate it behaves as expected; the tests are available in the sample project associated with this article.

By shifting this logic to the domain model, I've ensured that my controller's API method remains small and focused on its own concerns. As a result, it's easy to test that the controller does what it should, as there are relatively few paths through the method.

## Controller Responsibilities

Part of the controller's responsibility is to check ModelState and ensure it's valid. I'm doing this in the action method for clarity, but in a larger application I would eliminate this repetitive code within each action by using an Action Filter:

```
[HttpPost]
public IActionResult AddRsvp([FromBody]RsvpRequest rsvpRequest)
{
  if (!ModelState.IsValid)
  {
    return HttpBadRequest(ModelState);
  }
```

Assuming the ModelState is valid, the action must next fetch the appropriate Dinner instance using the identifier provided in the request. If the action can't find a Dinner instance matching that Id, it should return a Not Found result:

```
var dinner = _dinnerRepository.GetById(rsvpRequest.DinnerId);
if (dinner == null)
{
  return HttpNotFound("Dinner not found.");
}
```

Once these checks have been completed, the action is free to delegate the business operation represented by the request to the domain model, calling the AddRsvp method on the Dinner class that you saw previously, and saving the updated state of the domain model (in this case, the dinner instance and its collection of RSVPs) before returning an OK response:

```
  var result = dinner.AddRsvp(rsvpRequest.Name,
    rsvpRequest.Email,
    _systemClock.Now);

  _dinnerRepository.Update(dinner);
  return Ok(result);
}
```

Remember that I decided the Dinner class shouldn't have a dependency on the system clock, opting instead to have the current time passed into the method. In the controller, I'm passing in _systemClock.Now for the currentDateTime parameter. This is a local field that's populated via DI, which keeps the controller from being tightly coupled to the system clock, too. It's appropriate to use DI on the controller, as opposed to a domain entity, because controllers are always created by ASP.NET services containers; this will fulfill any dependencies the controller declares in its constructor. _systemClock is a field of type IDateTime, which is defined and implemented in just a few lines of code:

```
public interface IDateTime
{
  DateTime Now { get; }
}
public class MachineClockDateTime : IDateTime
{
  public DateTime Now { get { return System.DateTime.Now; } }
}
```

Of course, I also need to ensure the ASP.NET container is configured to use MachineClockDateTime whenever a class needs an instance of IDateTime. This is done in ConfigureServices in the Startup class, and in this case, although any object lifetime will work, I choose to use a Singleton because one instance of MachineClockDateTime will work for the whole application:

```
services.AddSingleton<IDateTime, MachineClockDateTime>();
```

With this simple abstraction in place, I'm able to test the controller's behavior based on whether the RSVP deadline has passed, and ensure the correct result is returned. Because I already have tests around the Dinner.AddRsvp method that verify it behaves as expected, I won't need very many tests of that same behavior through the controller to give me confidence that, when working together, the controller and domain model are working correctly.

Figure 2 **Business Logic in the Domain Model**

```
public RsvpResult AddRsvp(string name, string email, DateTime currentDateTime)
{
  if (currentDateTime > RsvpDeadlineDateTime())
  {
    return new RsvpResult("Failed - Past deadline.");
  }
  var rsvp = new Rsvp()
  {
    DateCreated = currentDateTime,
    EmailAddress = email,
    Name = name
  };
  if (MaxAttendees.HasValue)
  {
    if (Rsvps.Count(r => !r.IsWaitlist) >= MaxAttendees.Value)
    {
      rsvp.IsWaitlist = true;
      Rsvps.Add(rsvp);

      return new RsvpResult("Waitlist");
    }
  }
  Rsvps.Add(rsvp);

  return new RsvpResult("Success");
}
```

## Next Steps

Download the associated sample project to see the unit tests for Dinner and DinnersController. Remember that loosely coupled code is typically much easier to unit test than tightly coupled code riddled with "new" or static method calls that depend on infrastructure concerns. "New is glue" and the new keyword should be used intentionally, not accidentally, in your application. Learn more about ASP.NET Core and its support for dependency injection at docs.asp.net. ■

**Steve Smith** *is an independent trainer, mentor and consultant, as well as an ASP.NET MVP. He has contributed dozens of articles to the official ASP.NET Core documentation (docs.asp.net), and works with teams learning this technology. Contact him at ardalis.com or follow him on Twitter: @ardalis.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Doug Bunting*

# BEST FILE APIs

## Open Create Convert Print Save

files from your *applications*!


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

SCAN FOR
20% SAVINGS!

# BUSINESS FILE FORMATS


## ASPOSE.Cells

XLS, CSV, PDF, SVG, HTML, PNG
BMP, XPS, JPG, SpreadsheetML...

## ASPOSE.Words

DOC, RTF, PDF, HTML, PNG
ePUB, XML, XPS, JPG...

## ASPOSE.Pdf

PDF, XML, XSL-FO, HTML, BMP
JPG, PNG, ePUB...

## ASPOSE.Slides

PPT, POT, POTX, XPS, HTML
PNG, PDF...

## ASPOSE.BarCode

JPG, PNG, BMP, GIF, TIF, WMF
ICON...

## ASPOSE.Tasks

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

## ASPOSE.Email

MSG, EML, PST, EMLX
OST, OFT...

## ASPOSE.Imaging

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

+ MANY MORE!

Get your FREE evaluation copy at www.aspose.com

.NET | Java | Cloud

ASPOSE
Your File Format APIs

# Hosted Web Applications for the Enterprise

Tim Kulp

Developers who work in a structured enterprise know that it's not always easy to implement new technologies. Introducing something new to the enterprise like a Universal Windows Platform (UWP) app project can be a challenge when you don't have teams versed (or confident) in their ability to deliver this type of app. Many enterprises have a significant investment in their intranet Web applications and the teams to support them. Hosted Web applications (HWAs) provide a bridge for enterprise intranet Web apps to join the Windows Store for Business with little effort.

In this article, I'll convert an existing intranet Web application to a UWP app for the Windows Store and then enhance the app with native Windows functionalities. As shown in **Figure 1**, the Web application is a social recognition app called "Gotcha" that enables employees at Contoso Corp. to recognize their peers for acts of kindness or to show appreciation. Gotcha is meant to celebrate one another and build a stronger, happier team. This Web application is a great candidate for the Windows Store in that it provides easy integration with contacts, sharing and the camera.

This article discusses:

- Using UWP Bridges to bring Web apps to the Windows Store
- Building hosted Web applications in Visual Studio 2015
- Enhancing hosted Web applications with native functionality
- Working with other mobile platforms in the same hosted Web application

Technologies discussed:

Universal Windows Platform Apps, JavaScript, Web Development, Universal Windows Platform Bridges

## Bridges and Hosted Web Applications for the Enterprise

For enterprise developers, UWP Bridges and HWAs make sense because developers can maintain their existing tools, processes and deployment systems for their apps. The concept driving Bridges is to enable developers to bring their existing iOS, Android, Win32 or Web applications to the UWP and the cross-platform Windows Store. The goal is to enable developers to bring their existing code base to the UWP with the ability to light up the user's experience with features specific to Windows, such as Xbox achievements.

Bridges make bringing code to the UWP easy, but enterprise developers face other challenges that can make converting their code difficult. Developers in a corporation can face restrictions on what code editors are available to them, or where code can be deployed based on the sensitivity of the information within the application. The real value of UWP Bridges to the enterprise isn't the ease of converting apps, but the value from the ability to maintain existing enterprise software development tools, practices and change management to deliver UWP apps via the Windows Store.

As an example of this, HWAs (aka "Project Westminster") enable developers to embed Web applications into a UWP app. Once the

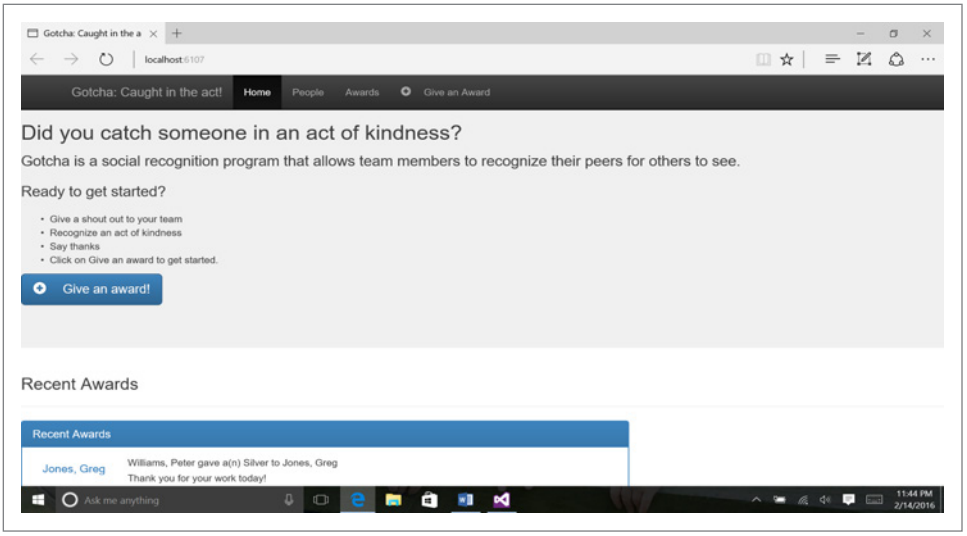



Figure 1 **Gotcha Web Application**

UWP app is built to use the HWA content, developers can use their existing practices to maintain the Web application, which will automatically update the Web application experience of the UWP app. Using the HWA Bridge, developers can focus on their Web application and include features specific to UWP through feature detection. Working with an existing intranet is a great use case for UWP Bridges, but developers outside the enterprise also can leverage HWAs to take their Web apps into the Windows Store. Any Web site can become an HWA as long as that site passes the Windows Store certification process.

The basics of HWAs are covered thoroughly in Kiril Seksenov's blog post, "Project Westminster in a Nutshell" (bit.ly/1PTxxW4). I recommend reading this post to get a deeper understanding of the ideas behind HWAs.

## Starting a Web HWA Project

To start converting the Gotcha Web application to an HWA, I create a new JavaScript UWP App in Visual Studio. Once the app is created, I open the package manifest and set the app's start page to be the URL of the Web application, as shown in **Figure 2**. In this case, the Gotcha Web application runs off of localhost:6107. With this start page set, the app will display the Web page at the specified address instead of the local content for the app.

Finally, set the content URIs in the package manifest. These URIs will tell your app what pages can access Windows Runtime (WinRT) libraries and what level of access those URIs can have. I've heard this described as defining where the app stops and the Internet starts, and I think of this as a great mental image of how to use content URIs. In the existing Gotcha Web application, there are clear distinctions between what the app does and what it can use the Internet to do. As an example, within Gotcha, users can share recognition to social networks like LinkedIn. The LinkedIn Web site isn't part of the Gotcha UWP app; it's part of the Internet. I would include only URIs that are directly used within the application and only those that need access to UWP APIs.

Using content URIs lets the developer protect the app's users by preventing access to WinRT libraries for unregistered URIs. For each URI, specify which level of access to WinRT libraries is needed:

- **All:** JavaScript code running on the Web application (in this case Gotcha) can access any and all UWP APIs that are defined through the App Capabilities.
- **Allow for Web Only:** JavaScript code running on the Web application can execute custom WinRT components included in the app package, but can't access all UWP APIs.
- **None:** JavaScript code running on the Web application can't access local UWP APIs (this is the default setting).

When setting content URIs, remember that this is a key component to application security for the UWP app user. Make sure to only provide the minimal permissions necessary to a URI for the necessary functions of the UWP app, as shown in **Figure 3**. Try to avoid setting the root URI WinRT Access to All if that's not truly needed within the app.

I don't recommend removing the existing content that comes with the project creation process (such as the .css, .js and WinJS

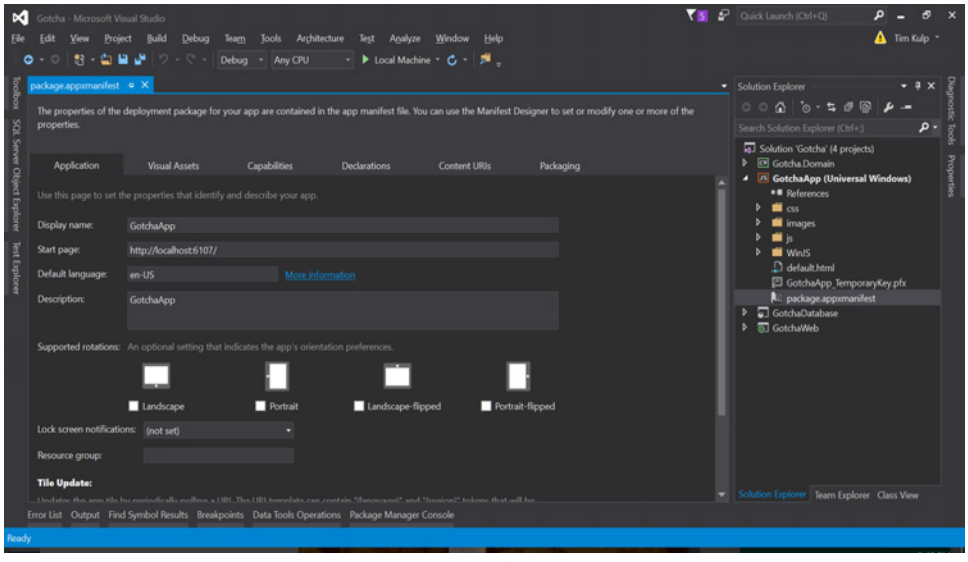



Figure 2 **Set the Start Page on the Package Manifest**

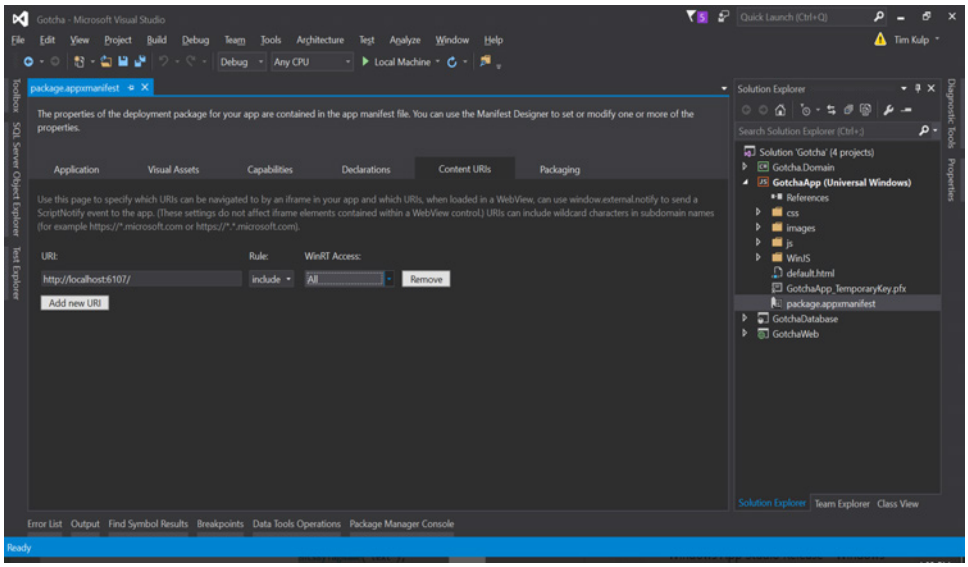



Figure 3 **Set Content URIs with the Least Privilege Needed to Run the App**

folders). This existing content gives developers an excellent framework that can be used to add local content to the Web HWA. Later in this article, I'll use these folders for local content to create some capabilities supplementing the Web HWA offline experience.

With the package manifest configured, run the UWP app. As shown in **Figure 4**, the Web application will now appear in an app window just like any other UWP app.

## Debugging a Hosted Web Application

HWAs have a slightly different debugging experience than a pure native app. With the UWP app open, press F12. This will show the F12 Developer Tools for the UWP app. Through the F12 Developer Tools I can debug, profile and test the Web application just like I can in a browser. While all the features of the F12 Developer Tools are useful for HWA developers, the components I've found most useful are debugging outside of Visual Studio and profiling network activity. I use these features to dig into a specific app's behaviors and understand any problems (like caching) that are providing an unexpected experience in the app.

Just like using F12 in the browser, developers can also alter the DOM and try out different UI experiences based on screen size or window size. This can be used to explore the layout changes of a Web application based on the responsive needs of the app. As an example, snapping the UWP app to display as a sidebar should reflow the app to present a great experience. If the resize doesn't reflow the app, F12 Developer Tools can be used to determine why not and experiment with the DOM to see what's needed to achieve the desired reflow.

## Adding Functionality for Your UWP App

With a basic HWA ready, I'm going to dive into the Web application's capabilities to enable some UWP APIs and light up the user's experience. Intranet Web applications can have some limitations that HWAs don't have. Using the UWP APIs, an HWA can have access to many of the local capabilities of the UWP device (such as camera, location and other sensors). To start, think about the use cases that are driving the implementation of the Web application to a hosted Web application. In the Gotcha app, I want users to be able to select from their Contacts who to give an award instead of just typing in the person's name and attaching a picture to the recognition.

To start, I create a remote GotchaNative.js code file on the Web application (the GotchaNative.js script file will be on the remote Web application server) that'll detect native API namespaces and execute the appropriate code to trigger our native use cases. Add the following code to the NativeGotcha.js code file:

```
var GotchaNative = (function () {
  if (typeof Windows != 'undefined') {
    // Add find contact here
    // Add capture photo here
  }});
```

This code builds the GotchaNative object, which will hold all the functionality of the UWP APIs. Centralizing these APIs allows for a single file to include on pages that will have added UWP API functionality. I isolate this file so that I can explicitly declare content URIs that include this specific file as URIs with access to the

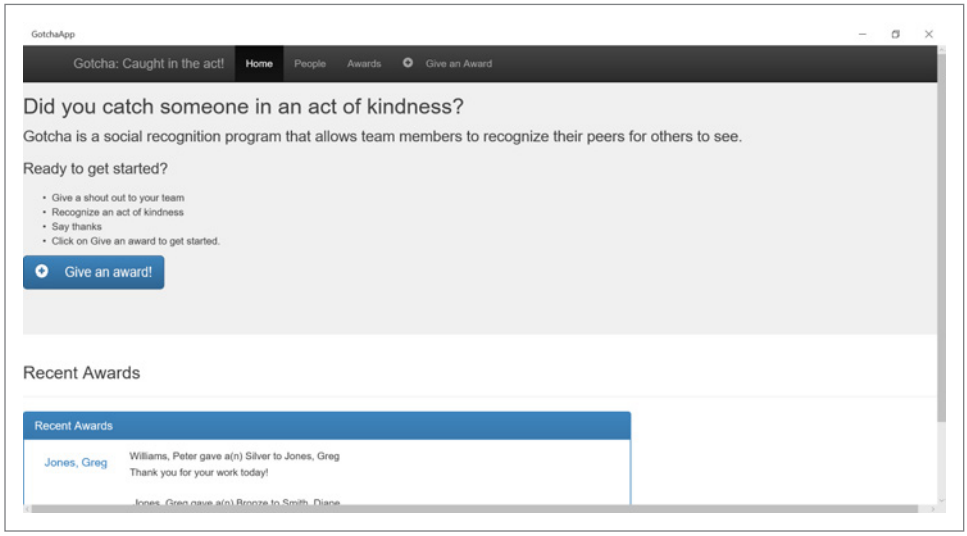



Figure 4 **Gotcha App Ready for the Windows Store**

# DEVELOPED FOR INTUITIVE USE

## DynamicPDF—Comprehensive PDF Solutions for .NET Developers

ceTe Software's DynamicPDF products provide real-time PDF generation, manipulation, conversion, printing, viewing, and much more. Providing the best of both worlds, the object models are extremely flexible but still supply the rich features you need as a developer. Reliable and efficient, the high-performance software is easy to learn and use. If you do encounter a question with any of our components, simply contact ceTe Software's readily available, industry-leading support team.

DynamicPDF

WWW.DYNAMICPDF.COM

**TRY OUR PDF SOLUTIONS FREE TODAY!**

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

ceTesoftware

needed UWP APIs. This way I can implement the security concept of least privilege and only give permission to URIs that require access to the UWP APIs.

Another benefit of isolating this file is to prepare for other native platforms, as well. I'll explore this later in the article but for now, consider this file to be the home for all native functionality that will be included in the Gotcha Web application.

## Extending Existing Functionality

Now with the GotchaNative object created, I can add the specific functionality to connect the HWA to the UWP APIs. For the first use case, the Gotcha Web application lets users enter in a person to recognize. In the UWP, users have the People app that stores contacts and would be much easier for the user to select than to type in the person's name. Replace the "Add find contact here" comment in the GotchaNative.js code file with the following:

```
this.FindContact = function () {
  var picker = new Windows.ApplicationModel.Contacts.ContactPicker();
  picker.desiredFieldsWithContactFieldType.append(
    Windows.ApplicationModel.Contacts.ContactFieldType.email);
  return picker.pickContactAsync();
}
```

This code is a basic connection to the ContactPicker API to select a contact from the list of contacts. A list of UWP APIs that can snap into Web HWAs can be found at rjs.azureWebsites.net. This Web site lists some of the more popular APIs with copy/paste-ready code to help build out a Web HWA project.

The functionality to add a person can be found on the Person View Model for the Gotcha Web site. Add the code in **Figure 5** to the Person View Model.

The code in **Figure 5** uses the FindContact method if the Windows object is defined due to the script running as an HWA. If the Windows object isn't defined, then the Web application will continue to use the existing Bootstrap modal window to collect information about the person to recognize. This code enables the Web application to use one approach to enter a person for recognition while the UWP app can leverage a different, more tailored, experience for the user.

Figure 5 **Code to Add to Person View Model**

```
if (Windows != undefined) {
  var nativeGotcha = new NativeGotcha();
  nativeGotcha.FindContact().done(function (contact) {
    if (contact !== null) {
      $("#txtWho").val(contact.displayName);
    } else {
      // Write out no contact selected
    }
  });
}
else {
  $('#add-person').on('shown.bs.modal', function () {
    $("#txtWho").focus();

}
```

Figure 6 **Feature Detection Code to Add to Home View Model**

```
$('#give-modal').on('shown.bs.modal', function () {
  if (typeof Windows != 'undefined') {
    var gotchaNative = new NativeGotcha();
    $("#btnCamera").click(function () {
      gotchaNative.CapturePicture().then(function (capturedItem) {
        // Do something with capturedItem;
      });
    }).show();
  }
  else {
    $("#btnCamera").hide();
  }
})
```

## Adding New Functionality

Sometimes enabling the UWP APIs will allow new functionality that would not normally exist in the Web application. In the Gotcha HWA, I want to enable users to take pictures and send those photos for recognition to other Gotcha users. This functionality would be new to the application and wouldn't exist in the Web application. When building an HWA, consider opportunities the UWP APIs open up for the app.

> In the Gotcha HWA, I want to enable users to take pictures and send those photos for recognition to other Gotcha users.

To start implementing this new functionality, first I replace the "Add capture photo" comment with the following code in the GotchaNative.js code file:

```
this.CapturePicture = function () {
  var captureUI = new Windows.Media.Capture.CameraCaptureUI();
  captureUI.photoSettings.format =
    Windows.Media.Capture.CameraCaptureUIPhotoFormat.png;
  return captureUI.captureFileAsync(
    Windows.Media.Capture.CameraCaptureUIMode.photo);
}
```

In this code, I open the capture UI, allow the user to take a picture and then return the picture to the caller. This new feature needs to be enabled on the Web application if the Windows object is defined. Using the feature detection like I did in the Person View Model, I add the code in **Figure 6** to the Home View Model, which is where the code is for adding the recognition.

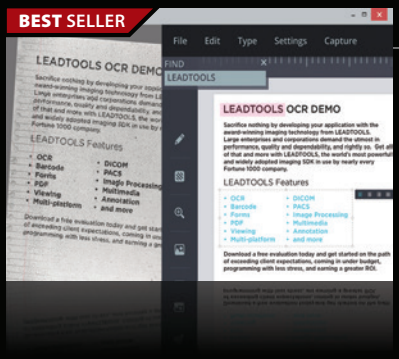
In the code in **Figure 6**, I hide the btnCamera if the Windows object is undefined. If the Windows object isn't undefined, then I configure the click event to trigger the CapturePicture function. For brevity, I've left a comment of what to do with the picture. With enabling the camera, I need to keep in mind that this HWA is still a UWP app and needs the Webcam capability enabled for the app, which can be done through the package manifest.

Enterprise Web applications can be converted to HWAs and leverage the UWP APIs of the Windows Store. Using existing Web applications makes it easy to build up UWP apps in the Windows Store, but HWAs assume an Internet connection. Next, I'll show you how to ensure a working app even without Internet access.

## Connecting Offline

The UWP app can have local files to provide an offline or local experience. Using local content lets you disconnect some of the UWP API calls from the hosted Web application. Like the feature

BEST SELLER



## LEADTOOLS Document Imaging SDKs V19 | from $2,995.00 SRP

LEAD TECHNOLOGIES

**Add powerful document imaging functionality to desktop, tablet, mobile & web applications.**

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

BEST SELLER

## DevExpress DXperience 15.2 | from $1,439.99

DevExpress

**The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.**

- WinForms Grid: New data-aware Tile View
- WinForms Grid & TreeList: New Excel-inspired Conditional Formatting
- .NET Spreadsheet: Grouping and Outline support
- ASP.NET: New Rich Text Editor-Word Processing control
- ASP.NET Reporting: New Web Report Designer

BEST SELLER

## Help & Manual Professional | from $586.04

ec software

**Help and documentation for .NET and mobile applications.**

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

BEST SELLER

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

ASPOSE
Your File Format APIs

**Every Aspose .NET component in one package.**

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

© 1996-2016 ComponentSource. All Rights Reserved. All prices correct at the time of press. Online prices may vary from those shown due to daily fluctuations & online discounts.

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R

detection that has already been done, linking to local content can be done through activating links. For Gotcha, I'll add a mapping feature that'll use the local mapping controls.

In the HWA, I can use the same feature detection that was used previously in the Home View Model. Using the right URI scheme will enable the HWA to jump between the Web content and local content. On the home screen I add the following link:

```
<a href="ms-appx:///default.html" id="lnkMap">
  <span class="glyphicon glyphicon-map-marker"> </span> Show Map
</a>
```

The link connects to the ms-appx:/// scheme, which lets the application connect to local content within the application package. When working within the ms-appx scheme, the app can connect to the UWP APIs necessary. This is similar to using the content URIs to declare what level of access a page has for access to APIs. Using ms-appx is like marking a URI as All. In this case, the default.html page has access to the UWP APIs. When the link appears within the HWA, users can click it to work within the app package.

For enterprise developers, this feature allows isolation of use cases that are specific to the UWP and not necessarily connected to the Web application. Connecting to package content also can allow you to avoid providing access to the UWP APIs for the Web application and keep all that access within the package.

## What About More Than One Store?

Depending on the enterprise, a Bring Your Own Device environment might exist. This means that an enterprise could already be targeting iOS or Android devices with existing apps. Android and iOS apps have concepts similar to HWAs called the WebView control. WebView lets developers provide a window within their app to an existing Web application and then interact with that Web application as if it were part of the native app (sound familiar?). HWAs can be built to play well with others, tailoring the functionality of an existing Web application to the platform delivering the Web app. Next, I'll update GotchaNative.js to support Android and show how HWA code would live side-by-side with JavaScript targeting other platforms.

Earlier, I set up GotchaNative.js, which is meant to hold all the native code in the app. The ViewModels will process the output of these native methods. To use APIs in any platform is similar to HWAs: First, the app must determine if the native APIs are available and then must call the appropriate API. To start updating GotchaNative.js, I'll add a property that tells me what native platform (if any) is recognized. For the following examples, I'm assuming that the Android app is using a WebView that declares "Android" as the local script namespace:

```
this.Platform = function () {
  if (Windows != 'undefined')
    return "Windows";
  else if (Android != 'undefined')
    return "Android";
  else
    return "Web";
}
```

This function allows my code to determine which platform it's going to be working with so that I can tailor code and response to the platform. From here, each method within GotchaNative.js can check for the platform to know how to proceed, as shown in **Figure 7**.

Now in the ViewModels, the code would be updated to use the platform detection to know what to do with the output of the GotchaNative methods. The following is an update to the camera button event from the HomeViewModel:

```
$("#btnCamera").click(function () {
  switch (gotchaNative.Platform()) {
    case "Windows":
      gotchaNative.CapturePicture().then(function (capturedItem) {
        // Do something with capturedItem;
      });
      break;
    case "Android":
      // Handle Android output
      break;
  }
}).show();
```

Now, as the app supports more platforms, I have a single place in my code to grow the capabilities for all platforms. If a new hot OS comes out tomorrow, the development team will be able to adapt the code to the new platform.

## Wrapping Up

Enterprise developers face numerous challenges in their work. Standardized processes and tools can be seen as a barrier to implementing new technologies like UWP apps. Using UWP Bridges, enterprise developers can bring their existing intranet or external Web apps to the Windows Store (either public or Windows Store for Business). Bridges for HWAs can turn a Web app into a cross-platform UWP app, leveraging the UWP APIs to light up the user's experience.

In this article, I explored how to take an existing Web application and convert it to an HWA while adding UWP app-specific features like accessing the camera and contacts. Using the ms-appx scheme, I showed how to jump from your hosted Web application to local content within the app package. Finally, I showed, with some planning, how HWAs can be built with many different platforms in mind, allowing the enterprise to keep up with the ever-expanding world of mobile devices. Using these techniques, HWAs can leverage the existing investments in Web applications, helping enterprises join the Windows Store ecosystem. ■

Figure 7 **GotchaNative.js Methods Checking for the Platform**

```
this.CapturePicture = function () {
  switch (this.Platform()) {
    case "Windows":
      var captureUI = new Windows.Media.Capture.CameraCaptureUI();
      captureUI.photoSettings.format =
        Windows.Media.Capture.CameraCaptureUIPhotoFormat.png;
      return captureUI.captureFileAsync(
        Windows.Media.Capture.CameraCaptureUIMode.photo);
      break;
    case "Android":
      Android.CaptureFile(); // Calls Android code in Native app
      break;
    default:
      return null;
      break;
}
```

**Tim Kulp** *is a senior technical architect living in Baltimore, Md. He is a Web, mobile and UWP developer, as well as author, painter, dad and "wannabe Mad Scientist Maker." Find him on Twitter: @seccode or via LinkedIn: linkedin.com/in/timkulp.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *John-David Dalton and Kiril Seksenov*


R#
ReSharper
THE LEGENDARY EXTENSION
TO VISUAL STUDIO*
jetbrains.com/resharper
JET
BRAINS
THE DRIVE
TO DEVELOP
* WARNING! PROLONGED USE
MAY CAUSE ADDICTION

# Maximize Your Model-View-ViewModel Experience with Roslyn

Alessandro Del Sole

Model-View-ViewModel (MVVM) is a very popular architectural pattern that works perfectly with XAML application platforms such as Windows Presentation Foundation (WPF) and the Universal Windows Platform (UWP). Architecting an application using MVVM provides, among many others, the benefit of clean separation between the data, the application logic and the UI. This makes applications easier to maintain and test, improves code re-use and enables designers to work against the UI without interacting with the logic or the data. Over the years, a number of libraries, project templates, and frameworks, such as Prism and the MVVM Light Toolkit, have been built to help developers implement MVVM more easily and efficiently. However, in some situations, you can't rely on external libraries, or you might simply want to be able to implement the pattern quickly while keeping focus on your code. Though there are a variety of MVVM implementations, most share a number of common objects whose generation can be easily automated with the Roslyn APIs. In this article, I'll explain how to create custom Roslyn refactorings that make it easy to generate elements common to every MVVM implementation. Because giving you a complete summary about MVVM isn't possible here, I'm assuming you already have basic knowledge of the MVVM pattern, of the related terminology and of the Roslyn code analysis APIs. If you need a refresher, you can read the following articles: "Patterns—WPF Apps with the Model-View-ViewModel Design Pattern" (msdn.com/magazine/dd419663), "C# and Visual Basic: Use Roslyn to Write a Live Code Analyzer for Your API" (msdn.com/magazine/dn879356), and "C#—Adding a Code Fix to Your Roslyn Analyzer" (msdn.com/magazine/dn904670).

The accompanying code is available in both C# and Visual Basic versions. The listings here are in C#; the Visual Basic listings are included in the online version of the article.

**This article discusses:**

- Creating a project for Roslyn refactorings
- Parsing common MVVM classes with Roslyn
- Making a refactoring available to specific platforms
- Generating a custom ViewModel with Roslyn
- Using a sample refactoring from Microsoft

**Technologies discussed:**

Roslyn, C#, Visual Basic

**Code download available at:**

msdn.com/magazine/0416magcode

## Common MVVM Classes

Any typical MVVM implementation requires at least the following classes (in some cases with slightly different names, depending on the MVVM flavor you apply):

**ViewModelBase** A base abstract class exposing members that are common to every ViewModel in the application. Common

Figure 1 **ViewModelBase Class (C#)**

```
abstract class ViewModelBase : System.ComponentModel.
INotifyPropertyChanged
{
  public event System.ComponentModel.PropertyChangedEventHandler PropertyChanged;

  // Raise a property change notification
  protected virtual void OnPropertyChanged(string propertyName)
  {
    PropertyChanged?.Invoke(this,
      new System.ComponentModel.PropertyChangedEventArgs(propertyName));
  }
}
```

members may vary depending on the application's architecture, but its most basic implementation brings change notification to any derived ViewModel.

**RelayCommand** A class that represents a command through which ViewModels can invoke methods. There are typically two flavors of RelayCommand, one generic and one non-generic. In this article I'll use the generic flavor (RelayCommand<T>).

I'm assuming you're already familiar with both, so I won't go into more detail. **Figure 1** represents the C# code for ViewModelBase.

This is the most basic implementation of ViewModelBase; it just provides property change notification based on the INotifyPropertyChanged interface. Of course, you might have additional members depending on your specific needs. **Figure 2** shows the C# code for RelayCommand<T>.

Figure 2 **RelayCommand<T> Class (C#)**

```
class RelayCommand<T> : System.Windows.Input.ICommand
{
  readonly Action<T> _execute = null;
  readonly Predicate<T> _canExecute = null;

  public RelayCommand(Action<T> execute)
    : this(execute, null)
  {
  }

  public RelayCommand(Action<T> execute, Predicate<T> canExecute)
  {
    if (execute == null)
        throw new ArgumentNullException(nameof(execute));

    _execute = execute;
    _canExecute = canExecute;
  }

  [System.Diagnostics.DebuggerStepThrough]
  public bool CanExecute(object parameter)
  {
    return _canExecute == null ? true : _canExecute((T)parameter);
  }

  public event EventHandler CanExecuteChanged;

  public void RaiseCanExecuteChanged()
  {
    var handler = CanExecuteChanged;
    if (handler != null)
    {
      handler(this, EventArgs.Empty);
    }
  }

  public void Execute(object parameter)
  {
    _execute((T)parameter);
  }
}
```

This is the most common implementation of RelayCommand<T> and it's appropriate for most MVVM scenarios. It's worth mentioning that this class implements the System.Windows.Input.ICommand interface, which requires implementing a method called CanExecute, whose goal is telling the caller if a command is available for execution.

## How Roslyn Can Simplify Your Life

If you don't work with external frameworks, Roslyn can be a real life-saver: You can create custom code refactorings that replace a class definition and automatically implement a required object, and you can easily automate the generation of ViewModel classes based on the model properties. **Figure 3** shows an example of what you'll achieve by the end of the article.

The benefit of this approach is that you always keep your focus on the code editor and you can implement the required objects very quickly. Moreover, you can generate a custom ViewModel based on the model class, as demonstrated later in the article. Let's start by creating a refactoring project.

## Creating a Project for Roslyn Refactorings

The first step is to create a new Roslyn refactoring. To accomplish this, you use the Code Refactoring (VSIX) project template, available in the Extensibility node under the language of your choice in the New Project dialog. Call the new project MVVM_Refactoring.

Click OK when ready. When Visual Studio 2015 generates the project, it automatically adds a class called MVVMRefactoringCodeRefactoringProvider, defined inside the CodeRefactoringProvider.cs (or .vb for Visual Basic) file. Rename the class and the file to MakeViewModelBaseRefactoring and MakeViewModelBaseRefactoring.cs, respectively. For the sake of clarity, remove both the auto-generated ComputeRefactoringsAsync and ReverseTypeNameAsync methods (the latter is auto-generated for demonstration purposes).

## Investigating a Syntax Node

As you might know, the main entry point for a code refactoring is the ComputeRefactoringsAsync method, which is responsible for creating a so-called quick action that will be plugged into the code editor's light bulb, if code analysis of a syntax node satisfies the required rules. In this particular case, the ComputeRefactoringsAsync method must detect if the developer is invoking the light bulb over a class declaration. With the help of the Syntax Visualizer tool window, you can easily understand the syntax elements with which you need to work. More specifically, in C# you must detect whether the syntax node is a ClassDeclaration, represented by an object of type Microsoft.CodeAnalysis.CSharp.Syntax.ClassDeclarationSyntax (see **Figure 4**), while in Visual Basic you determine if the syntax node is a ClassStatement, represented by an object of type Microsoft.CodeAnalysis.VisualBasic.Syntax.ClassStatementSyntax. Actually, in Visual Basic, ClassStatement is a child node of ClassBlock, which represents the entire code for a class. The reason C# and Visual Basic have different objects is because of the way each represents a class definition: C# uses the class keyword with curly braces as delimiters, and Visual Basic uses the Class keyword with the End Class statement as a delimiter.

## Creating an Action

The first code refactoring I'll discuss relates to the ViewModelBase class. The first step is to write the ComputeRefactoringsAsync method in the MakeViewModelBaseRefactoring class. With this method, you check if the syntax node represents a class declaration; if so, you can create and register an action that will be available in the light bulb. **Figure 5** shows how to accomplish this in C# (see inline comments).

With this code, you've registered an action that can be invoked on the syntax node if this is a class declaration. The action is performed by the MakeViewModelBaseAsync method, which implements the refactoring logic and provides a brand new class.

## Code Generation

Roslyn not only provides an object-oriented, structured way to represent source code, it also allows parsing source text and generating a syntax tree with full fidelity. To generate a new syntax tree from pure text, you invoke the SyntaxFactory.ParseSyntaxTree method. It takes an argument of type System.String that contains the source code from which you want to generate a SyntaxTree.

Roslyn also offers the VisualBasicSyntaxTree.ParseText and CSharpSyntaxTree.ParseText methods to accomplish the same result; however, in this case it makes sense to use SyntaxFactory.ParseSyntaxTree because the code invokes other Parse methods from SyntaxFactory, as you'll see shortly.

Once you have a new SyntaxTree instance, you can perform code analysis and other code-related operations on it. For example, you can parse the source code of a whole class, generate a syntax tree from it, replace a syntax node in the class and return a new class. In the case of the MVVM pattern, common classes have a fixed structure, so the process of parsing source text and replacing a class definition with a new one is fast and easy. By taking advantage of so-called multi-line string literals, you can paste a whole class definition into an object of type System.String, then get a SyntaxTree from it, retrieve the SyntaxNode that corresponds to the class definition and replace the original class in the tree with the new one. I'll first demonstrate how to accomplish this with regard to the ViewModelBase class. More specifically, **Figure 6** shows the code for C#.

Because the SyntaxFactory type is used many times, you could consider doing a static import and, thus, simplify your code by adding an Imports Microsoft.CodeAnalisys.VisualBasic.SyntaxFactory directive in Visual Basic and a using static Microsoft.CodeAnalysis.CSharp.SyntaxFactory directive in C#. There's no static import here to make it easier to discover some of the methods SyntaxFactory offers.

Notice that the MakeViewModelBaseAsync method takes three arguments:

- A Document, which represents the current source code file
- A ClassDeclarationSyntax (in Visual Basic, it's ClassStatementSyntax), which represents the class declaration over which code analysis is executed
- A CancellationToken, which is used in case the operation must be canceled

The code first invokes SyntaxFactory.ParseSyntaxTree to get a new SyntaxTree instance based on the source text that represents the

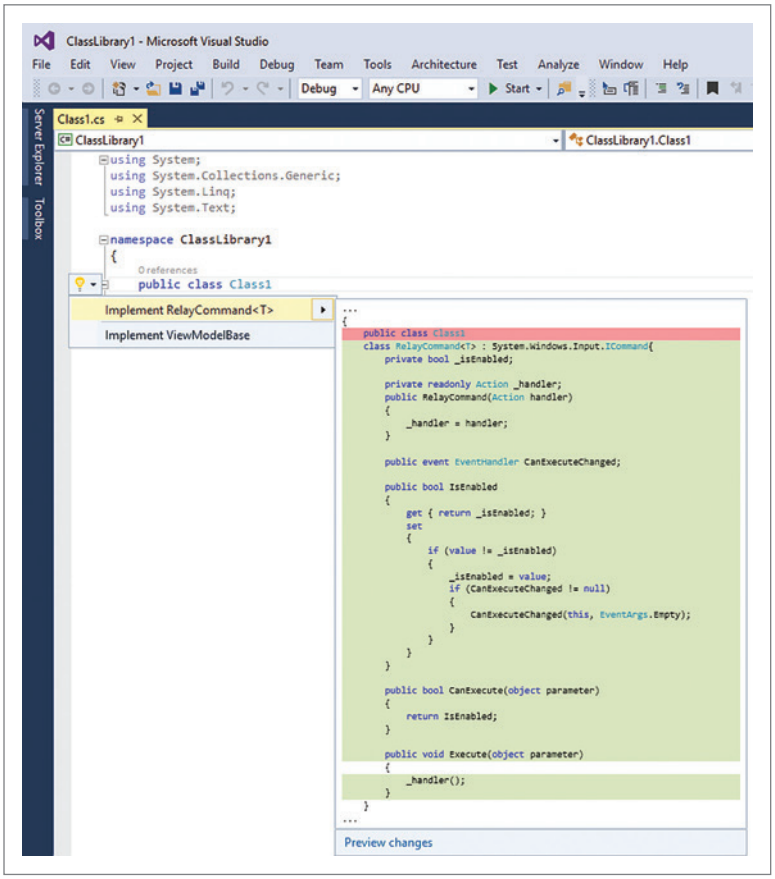



Figure 3 **Implementing MVVM Objects with a Custom Roslyn Refactoring**

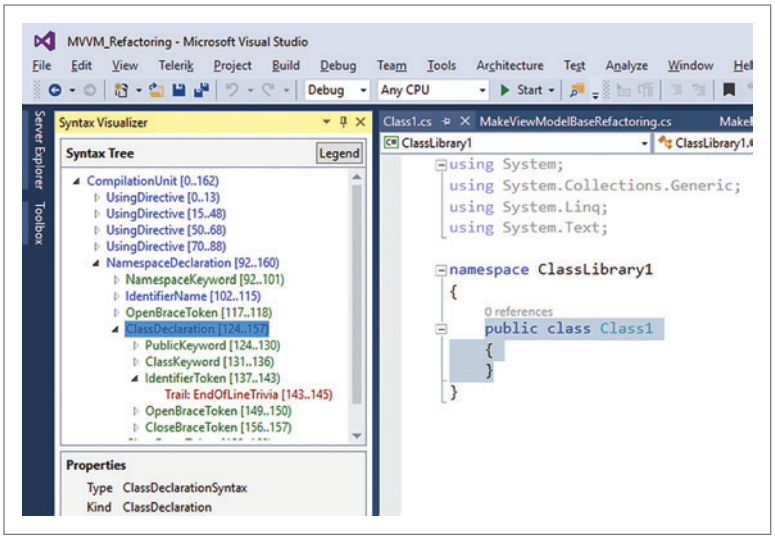



Figure 4 **Understanding a Class Declaration**

# Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/boston

## Boston JUNE 13-16

HYATT REGENCY, CAMBRIDGE, MA

# BETTER CODE FOR ALL

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

Boston

BOSTON 2016 CAMPAIGN FOR CODE

## PRACTICAL & UNBIASED TRAINING FOR DEVELOPERS:

- ALM / DevOps
- Cloud Computing
- Database and Analytic
- Mobile Client
- Software Practices
- UX / Design
- Visual Studio / .NET Framework
- Web Client
- Web Server

## Register by May 18 and Save $200

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

USE PROMO CODE VSLMAYTI

PLATINUM SPONSOR

Microsoft Virtual Academy
LEARNING HAPPENS HERE

SUPPORTED BY

Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

VSLIVE.COM/BOSTON

# Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/austin

## Austin MAY 16-19

HYATT REGENCY, AUSTIN, TX


### PRACTICAL & UNBIASED TRAINING FOR DEVELOPERS:

- ALM / DevOps
- ASP.NET
- Cloud Computing
- Database and Analytics
- JavaScript / HTML5 Client
- Mobile Client
- Software Practices
- UX / Design
- Visual Studio / .NET Framework
- Windows Client (Windows 10/UWP, WPF)

**Sessions are filling up quickly**

**REGISTER TODAY!**

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

**USE PROMO CODE VSLMAYTI**

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

# YOUR Austin CODE COUNTS

AUSTIN 2016 CAMPAIGN FOR CODE

PLATINUM SPONSOR

Microsoft Virtual Academy
LEARNING HAPPENS HERE

GOLD SPONSOR

Octopus Deploy

SUPPORTED BY

Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

**VSLIVE.COM/AUSTIN**

ViewModelBase class. The invocation of GetRoot is required to get the root SyntaxNode instance for the syntax tree. In this particular scenario, you know in advance that the parsed source text has only one class definition, so the code invokes FirstOrDefault<T> over OfType<T> to retrieve the one descendant node of the required type, which is ClassDeclarationSyntax in C# and ClassBlockSyntax in Visual Basic. At this point, you need to replace the original class definition with the ViewModelBase class. To accomplish this, the code first invokes Document.GetSyntaxRootAsync to asynchronously retrieve the root node for the document's syntax tree, then it invokes ReplaceNode to replace the old class definition with the new ViewModelBase class. Notice how the code detects if a using (C#) or Imports (Visual Basic) directive exists for the System.ComponentModel namespace by investigating the CompilationUnitSyntax.Usings and the CompilationUnitSyntax.Imports collections, respectively. If not, a proper directive is added. This is useful for adding a directive at the code file level if not already available.

Remember, in Roslyn, objects are immutable. This is the same concept that applies to the String class: You actually never modify a string, so when you edit a string or invoke methods like Replace, Trim or Substring, you get a new string with the specified changes. For this reason, every time you need to edit a syntax node, you actually create a new syntax node with updated properties.

In Visual Basic, the code also needs to retrieve the parent ClassBlockSyntax for the current syntax node, which is instead of type ClassStatementSyntax. This is required to retrieve the instance of the actual SyntaxNode that will be replaced. Providing a common implementation of the RelayCommand<T> class works exactly the same, but you need to add a new code refactoring. To accomplish this, right-click the project name in Solution Explorer, then select Add | New Item. In the Add New Item dialog, select the Refactoring template and name the new file MakeRelayCommandRefactoring.cs (or .vb, for Visual Basic). The refactoring logic is the same as for the ViewModelBase class (of course, with different source text). **Figure 7** shows the full C# code for the new refactoring, which includes the ComputeRefactoringsAsync and MakeRelayCommandAsync methods.

You've successfully completed two custom refactorings and you now have the basics to implement additional refactorings, depending on your implementation of the MVVM pattern (such as a message broker, a service locator and service classes).

As an alternative, you could use the SyntaxGenerator class. This offers language-agnostic APIs, which means the code you write results in a refactoring that targets both Visual Basic and C#. However, this approach requires generating every single syntax element for the source text. By using SyntaxFactory.ParseSyntaxTree, you can parse any source text. This is especially useful if you write developer tools that need to manipulate source text that you don't know in advance.

Figure 5 **The Main Entry Point: ComputeRefactoringsAsync Method (C#)**

```
private string Title = "Make ViewModelBase class";
public async sealed override Task
ComputeRefactoringsAsync(CodeRefactoringContext context)
{

  // Get the root node of the syntax tree
  var root = await context.Document.
    GetSyntaxRootAsync(context.CancellationToken).
    ConfigureAwait(false);

  // Find the node at the selection.
  var node = root.FindNode(context.Span);

  // Is this a class statement node?
  var classDecl = node as ClassDeclarationSyntax;
  if (classDecl == null)
  {
    return;
  }

  // If so, create an action to offer a refactoring
  var action = CodeAction.Create(title: Title,
    createChangedDocument: c =>
    MakeViewModelBaseAsync(context.Document,
      classDecl, c), equivalenceKey: Title);

  // Register this code action.
  context.RegisterRefactoring(action);
}
```

Figure 6 **MakeViewModelBaseAsync: Generating a New Syntax Tree from Source Text (C#)**

```
private async Task<Document> MakeViewModelBaseAsync(Document document,
  ClassDeclarationSyntax classDeclaration, CancellationToken cancellationToken)
{

  // The class definition represented as source text
  string newImplementation = @"abstract class ViewModelBase : INotifyPropertyChanged
{
public event System.ComponentModel.PropertyChangedEventHandler PropertyChanged;

// Raise a property change notification
protected virtual void OnPropertyChanged(string propertyName)
{
  PropertyChanged?.Invoke(this, new System.ComponentModel.
    PropertyChangedEventArgs(propertyName));
}
}
";
  // 1. ParseSyntaxTree() gets a new SyntaxTree from the source text
  // 2. GetRoot() gets the root node of the tree
  // 3. OfType<ClassDeclarationSyntax>().FirstOrDefault()
  //    retrieves the only class definition in the tree
  // 4. WithAdditionalAnnotations() is invoked for code formatting
  var newClassNode = SyntaxFactory.ParseSyntaxTree(newImplementation).
    GetRoot().DescendantNodes().
    OfType<ClassDeclarationSyntax>().
    FirstOrDefault().
    WithAdditionalAnnotations(Formatter.Annotation, Simplifier.Annotation);

  // Get the root SyntaxNode of the document
  var root = await document.GetSyntaxRootAsync();

  // Generate a new CompilationUnitSyntax (which represents a code file)
  // replacing the old class with the new one
  CompilationUnitSyntax newRoot = (CompilationUnitSyntax)root.
    ReplaceNode(classDeclaration, newClassNode).NormalizeWhitespace();

  // Detect if a using System.ComponentModel directive already exists.
  if ((newRoot.Usings.Any(u => u.Name.ToFullString() ==
    "System.ComponentModel"))== false)
  {
    // If not, add one
    newRoot = newRoot.AddUsings(
      SyntaxFactory.UsingDirective(SyntaxFactory.
      QualifiedName(SyntaxFactory.IdentifierName("System"),
                    SyntaxFactory.IdentifierName("ComponentModel"))));
  }

  // Generate a new document based on the new SyntaxNode
  var newDocument = document.WithSyntaxRoot(newRoot);

  // Return the new document
  return newDocument;
}
```

## Availability: UWP Apps and WPF

Instead of making custom refactorings available universally, it makes sense to restrict their availability in the light bulb only to those platforms you actually use MVVM with, such as WPF and UWP. In the ComputeRefactoringsAsync method, you can get an instance of the semantic model for the current document and then invoke the GetTypeByMetadataName method from the Compilation property. For example, the following code demonstrates how to make the refactoring available only to UWP apps:

```
// Restrict refactoring availability to Windows 10 apps only
var semanticModel = await context.Document.GetSemanticModelAsync();
var properPlatform = semanticModel.Compilation.
  GetTypeByMetadataName("Windows.UI.Xaml.AdaptiveTrigger");

if (properPlatform != null)
{
  var root = await context.Document.
    GetSyntaxRootAsync(context.CancellationToken).
    ConfigureAwait(false);

  // ...
}
```

Because the Windows.UI.Xaml.AdaptiveTrigger type exists only in UWP apps, the refactoring will be available if the code analysis engine detects that this type has been referenced. If you wanted to make the refactoring available to WPF, you could write the following check:

```
// Restrict refactoring availability to WPF apps only
var semanticModel = await context.Document.GetSemanticModelAsync();
var properPlatform = semanticModel.Compilation.
  GetTypeByMetadataName("System.Windows.Navigation.JournalEntry");
```

Similarly, the System.Windows.Navigation.JournalEntry uniquely exists in WPF, so a non-null result from GetTypeByMetadataName

Figure 7 **Code Refactoring That Implements the RelayCommand<T> Class (C#)**

```
[ExportCodeRefactoringProvider(LanguageNames.CSharp,
  Name = nameof(MakeRelayCommandRefactoring)), Shared]
internal class MakeRelayCommandRefactoring : CodeRefactoringProvider
{
  private string Title = "Make RelayCommand<T> class";
  public async sealed override Task
    ComputeRefactoringsAsync(CodeRefactoringContext context)
  {
    var root = await context.Document.GetSyntaxRootAsync(context.
      CancellationToken).
        ConfigureAwait(false);

    // Find the node at the selection.
    var node = root.FindNode(context.Span);

    // Only offer a refactoring if the selected node is
    // a class statement node.
    var classDecl = node as ClassDeclarationSyntax;
    if (classDecl == null)
    {
      return;
    }
    var action = CodeAction.Create(title: Title,
      createChangedDocument: c =>
      MakeRelayCommandAsync(context.Document,
      classDecl, c), equivalenceKey: Title);

    // Register this code action.
    context.RegisterRefactoring(action);
  }

  private async Task<Document>
    MakeRelayCommandAsync(Document document,
    ClassDeclarationSyntax classDeclaration, CancellationToken cancellationToken)
  {

    // The class definition represented as source text
    string newImplementation = @"
class RelayCommand<T> : ICommand
{
  readonly Action<T> _execute = null;
  readonly Predicate<T> _canExecute = null;

  public RelayCommand(Action<T> execute)
    : this(execute, null)
  {
  }

  public RelayCommand(Action<T> execute, Predicate<T> canExecute)
  {
    if (execute == null)
      throw new ArgumentNullException(""execute"");

    _execute = execute;
      _canExecute = canExecute;
  }

  [System.Diagnostics.DebuggerStepThrough]
  public bool CanExecute(object parameter)
  {
    return _canExecute == null ? true : _canExecute((T)parameter);
  }

  public event EventHandler CanExecuteChanged;

  public void RaiseCanExecuteChanged()
  {
    var handler = CanExecuteChanged;
    if (handler != null)
    {
      handler(this, EventArgs.Empty);
    }
  }

  public void Execute(object parameter)
  {
    _execute((T)parameter);
  }
}
";
    // 1. ParseSyntaxTree() gets a new SyntaxTree from the source text
    // 2. GetRoot() gets the root node of the tree
    // 3. OfType<ClassDeclarationSyntax>().FirstOrDefault() retrieves the only class
    //      definition in the tree
    // 4. WithAdditionalAnnotations() is invoked for code formatting
    var newClassNode = SyntaxFactory.ParseSyntaxTree(newImplementation).
      GetRoot().DescendantNodes().
      OfType<ClassDeclarationSyntax>().
      FirstOrDefault().
      WithAdditionalAnnotations(Formatter.Annotation, Simplifier.Annotation);

    // Get the root SyntaxNode of the document
    var root = await document.GetSyntaxRootAsync(cancellationToken);

    // Generate a new CompilationUnitSyntax (which represents a code file)
    // replacing the old class with the new one
    CompilationUnitSyntax newRoot = (CompilationUnitSyntax) root.
      ReplaceNode(classDeclaration,
      newClassNode).NormalizeWhitespace();

    if ((newRoot.Usings.Any(u => u.Name.ToFullString() == "System.Windows.Input"))
      == false)
    {
      newRoot = newRoot.AddUsings(
        SyntaxFactory.UsingDirective(SyntaxFactory.
        QualifiedName(SyntaxFactory.IdentifierName("System"),
          SyntaxFactory.IdentifierName("Windows.Input"))));
    }

    // Generate a new document based on the new SyntaxNode
    var newDocument = document.WithSyntaxRoot(newRoot);

    // Return the new document
    return newDocument;
  }
}
```

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

# JOIN US on the CAMPAIGN TRAIL in 2016!

AUSTIN 2016 CAMPAIGN FOR CODE

**MAY 16 - 19**
HYATT AUSTIN, TX
**vslive.com/austin**
See pages 40 – 41 for more info

LAST CHANCE!

BOSTON 2016 CAMPAIGN FOR CODE

**JUNE 13 - 16**
HYATT CAMBRIDGE, MA
**vslive.com/boston**
See pages 50 – 51 for more info

REDMOND 2016 CAMPAIGN FOR CODE

**AUGUST 8 - 12**
MICROSOFT HQ, REDMOND, WA
**vslive.com/redmond**
See pages 56 – 57 for more info


ANAHEIM 2016 CAMPAIGN FOR CODE

**SEPTEMBER 26 - 29**
HYATT ORANGE COUNTY, CA – A DISNEYLAND® GOOD NEIGHBOR HOTEL
**vslive.com/anaheim**
See pages 72 – 73 for more info

WASHINGTON D.C. 2016 CAMPAIGN FOR CODE

**OCTOBER 3 - 6**
RENAISSANCE, WASHINGTON, D.C.
**vslive.com/dc**
See pages 74 – 75 for more info

PART OF LIVE! 360

ORLANDO 2016 CAMPAIGN FOR CODE

**DECEMBER 5 - 9**
LOEWS ROYAL PACIFIC ORLANDO, FL
**vslive.com/orlando**
See pages 64 – 65 for more info

CONNECT WITH VISUAL STUDIO LIVE!

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

VSLIVE.COM

means that code analysis is running against a WPF project. Of course, you could combine both checks to make the refactorings available to both platforms.

## Testing the Code

You can test the work you've done so far in the Visual Studio experimental instance by pressing F5. For example, create a WPF project and add this very simple class:

```
class EmptyClass
{
  string aProperty { get; set; }
}
```

Right-click the class declaration, then select Quick Actions from the context menu. At this point, the light bulb shows the two new refactorings as expected and provides the proper suggestion (see **Figure 3** for reference).

If you want to publish custom refactorings, the Code Refactoring (VSIX) project template automates the generation of a VSIX package that can be published to the Visual Studio Gallery. If you'd rather publish your work as a NuGet package, the trick is creating an Analyzer with the Code Fix project and then adding Code Fix item templates.

## Generating a Custom ViewModel with Roslyn

If you're wondering why using Roslyn might be a better approach just to add static text to a class, imagine you want to automate the generation of a ViewModel from a business class, which is the model. In this case, you can generate a new ViewModel class and add the necessary properties based on the data exposed by the model. Just to give you an idea, **Figure 8** shows how to produce a refactoring called Make ViewModel class, which demonstrates how to create a simplified version of a ViewModel.

This code generates a ViewModel class that exposes an ObservableCollection of the model type, plus an empty constructor where you should implement your logic, as shown in **Figure 9**. Of course, this code should be extended with any additional members you might need, such as data properties and commands, and should be improved with a more efficient pluralization algorithm.

## Learn from Microsoft: The INotifyPropertyChanged Refactoring

One of the repetitive tasks you do with MVVM is implementing change notification to classes in your data model via the System.ComponentModel.INotifyPropertyChanged interface. For instance, if you had the following Customer class:

```
class Customer
{
  string CompanyName { get; set; }
}
```

You should implement INotifyPropertyChanged so that bound objects are notified of any changes to data:

```
class Customer : INotifyPropertyChanged
{
  public event PropertyChangedEventHandler PropertyChanged;
  string _companyName;
  string CompanyName
  {
    get { return _companyName; }
    set {
      _companyName = value;
      PropertyChanged?.Invoke(this,
        new PropertyChangedEventArgs(nameof(CompanyName))); }
  }
}
```

Figure 8 **Generating a ViewModel Based on a Model**

```
private async Task<Document> MakeViewModelAsync(Document document,
  ClassDeclarationSyntax classDeclaration, CancellationToken cancellationToken)
{
  // Get the name of the model class
  var modelClassName = classDeclaration.Identifier.Text;
  // The name of the ViewModel class
  var viewModelClassName = $"{modelClassName}ViewModel";

  // Only for demo purposes, pluralizing an object is done by
  // simply adding the "s" letter. Consider proper algorithms
  string newImplementation = $@"class {viewModelClassName} : INotifyPropertyChanged
{{
public event PropertyChangedEventHandler PropertyChanged;

// Raise a property change notification
protected virtual void OnPropertyChanged(string propname)
{{
  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propname));
}}

private ObservableCollection<{modelClassName}> _{modelClassName}s;

public ObservableCollection<{modelClassName}> {modelClassName}s
{{
  get {{ return _{modelClassName}s; }}
  set
  {{
    _{modelClassName}s = value;
    OnPropertyChanged(nameof({modelClassName}s));
  }}
}}

public {viewModelClassName}() {{
// Implement your logic to load a collection of items
}}
}}
";

    var newClassNode = SyntaxFactory.ParseSyntaxTree(newImplementation).
      GetRoot().DescendantNodes().
      OfType<ClassDeclarationSyntax>().
      FirstOrDefault().
      WithAdditionalAnnotations(Formatter.Annotation, Simplifier.Annotation);

    // Retrieve the parent namespace declaration
    var parentNamespace = (NamespaceDeclarationSyntax) classDeclaration.Parent;
    //Add the new class to the namespace
    var newParentNamespace =
      parentNamespace.AddMembers(newClassNode).NormalizeWhitespace();

    var root = await document.GetSyntaxRootAsync(cancellationToken);

    CompilationUnitSyntax newRoot = (CompilationUnitSyntax)root;

    newRoot = newRoot.
              ReplaceNode(parentNamespace, newParentNamespace).NormalizeWhitespace();

    newRoot = newRoot.AddUsings(SyntaxFactory.UsingDirective(SyntaxFactory.
      QualifiedName(SyntaxFactory.IdentifierName("System"),
        SyntaxFactory.IdentifierName("Collections.ObjectModel"))),
        SyntaxFactory.UsingDirective(SyntaxFactory.
      QualifiedName(SyntaxFactory.IdentifierName("System"),
        SyntaxFactory.IdentifierName("ComponentModel"))));

    // Generate a new document based on the new SyntaxNode
    var newDocument = document.WithSyntaxRoot(newRoot);

    // Return the new document
    return newDocument;
  }
```

# Microsoft Cloud Training for the Enterprise

We bring the cloud experts. You choose your pace.

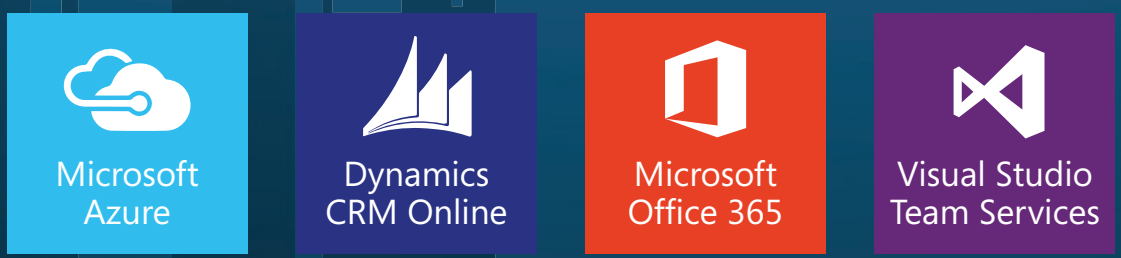
**On-Demand Online Training**

Choose the time, the place and the pace to fit your schedule.

**Instructor-Led Training**

Engage in real time with your instructor either online or in-person for a personalized training experience. (Onsite or Virtual)

Learning Paths Include:




Go to Opsgility.com now to start your free, 30-day trial with code: MSDNMAG

**REAL CODE. REAL LABS. REAL LEARNING.**

**Opsgility.com**

opsgility

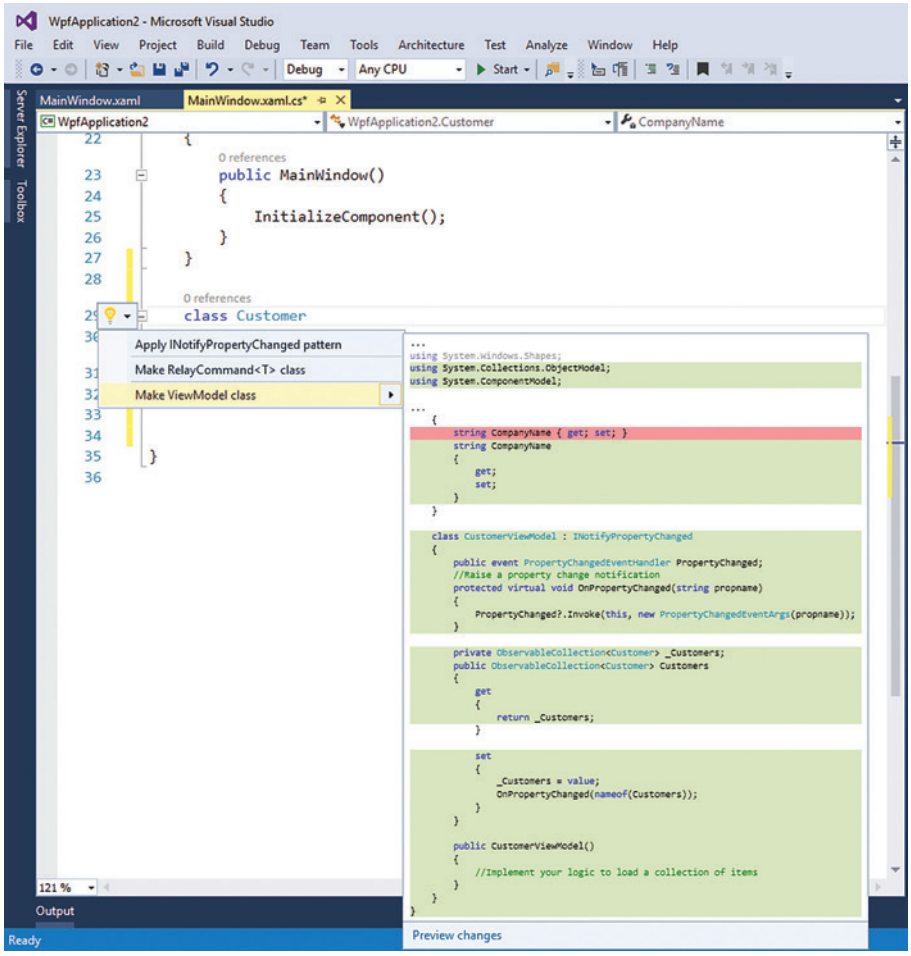



Figure 9 **Automating the Generation of a ViewModel Class**

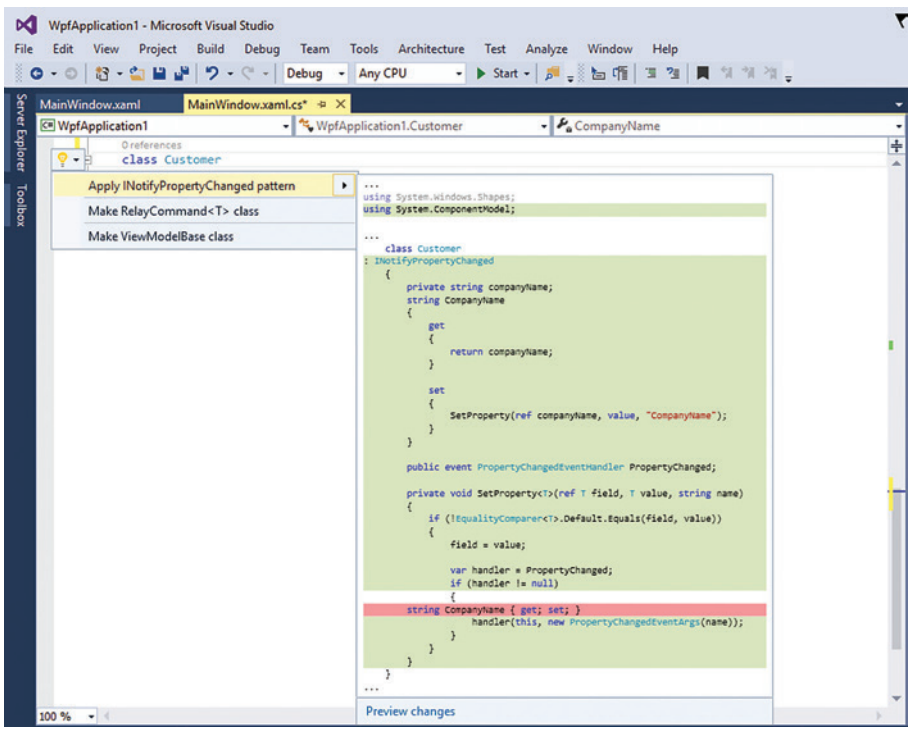



Figure 10 **Implementing the INotifyPropertyChanged Interface**

As you can imagine, in a data model made of many classes with dozens of properties, this task can take a lot of time. Among the companion samples for Roslyn, Microsoft shipped a code refactoring that automates the implementation of the INotifyPropertyChanged interface with a simple click. It's called ImplementNotifyPropertyChanged and is available for both C# and Visual Basic in the Src/Samples subfolder of the Roslyn repository at github.com/dotnet/roslyn. If you compile and test the example, you'll see how fast and efficient it makes implementing the INotifyPropertyChanged interface, as shown in **Figure 10**.

This example is particularly useful because it shows how to use the Roslyn APIs to walk through an object definition, how to parse specific members and how to make edits to existing properties without supplying a completely new class definition. Studying the source code for this example is definitely recommended in order to understand more complex code generation scenarios.

## Wrapping Up

Among its almost infinite number of possible uses, Roslyn also makes it incredibly easy to support the Model-View-ViewModel pattern. As I showed in this article, you can leverage the Roslyn APIs to parse the source code of certain classes required in any MVVM implementation, such as ViewModelBase and RelayCommand<T>, and generate a new syntax node that can replace an existing class definition. And Visual Studio 2015 will show a preview in the light bulb, providing another amazing coding experience. ■

**Alessandro del Sole** *has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos and articles about .NET development with Visual Studio. Del Sole works as a Solution Developer Expert for Brain-Sys (brain-sys.it), focusing on .NET development, training and consulting. You can follow him on Twitter: @progalex.*

**Thanks** to the following technical experts for reviewing this article: *Jason Bock (Magenic) and Anthony Green (Microsoft)*

FREE TRIAL

# DocuVieware

## Universal HTML5 Viewer & Document Management Kit

supports nearly 100 file formats

- zero-footprint solution
- super-easy integration
- fast & crystal clear rendering
- fully customizable UI look & feel
- mobile devices optimization
- annotations, thumbnails bookmarks & text search

## HOT FEATURES IN THE NEW MAJOR VERSION

- Office Open XML (.docx) support
- TWAIN acquisition
- PDF form fields interactions
- Custom snap-ins
- Higher speed and lower memory usage

Download the **Free Trial** and check the **Online Demos** at www.docuvieware.com

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

VSLIVE.COM/AUSTIN

# YOUR Austin CODE COUNTS

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

AUSTIN, TX
MAY 16-19, 2016
THE HYATT REGENCY

**VISUAL STUDIO LIVE!** is bringing back its unique brand of practical, unbiased, Developer training to the deep heart of Texas. We're all set to convene in Austin this May, where your code not only counts, it's crucial! From May 16 – 19, we're offering four days of sessions, workshops and networking events—all designed to help you elevate your code-writing abilities to write winning applications across all platforms.

## Get out and code.
## Register to join us today!

SESSIONS ARE FILLING UP QUICKLY—
REGISTER TODAY!

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

USE PROMO CODE VSLMAY2

PLATINUM SPONSOR
Microsoft Virtual Academy
LEARNING HAPPENS HERE

GOLD SPONSOR
Octopus Deploy

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

# AUSTIN AGENDA AT-A-GLANCE

ALM / DevOps | ASP.NET | Cloud Computing | Database and Analytics | JavaScript / HTML5 Client | Mobile Client | Software Practices | UX / Design | Visual Studio / .NET | Windows Client

| START TIME | END TIME | Visual Studio Live! Pre-Conference Workshops: Monday, May 16, 2016 *(Separate entry fee required)* | | |
|---|---|---|---|---|
| 9:00 AM | 6:00 PM | **M01** Workshop: DevOps in a Day - *Brian Randell* | **M02** Workshop: SQL Server for Developers - *Andrew Brust and Leonard Lobel* | **M03** Workshop: Building Modern Mobile Apps - *Brent Edwards and Kevin Ford* |
| 6:45 PM | 9:00 PM | Dine-A-Round | | |

| START TIME | END TIME | Visual Studio Live! Day 1: Tuesday, May 17, 2016 | | | |
|---|---|---|---|---|---|
| 8:00 AM | 9:00 AM | **KEYNOTE: Visual Studio: Looking into the Future**, *Tarek Madkour, Principal Group Program Manager, Microsoft* | | | |
| 9:15 AM | 10:30 AM | **T01** ASP.NET Core 1.0 in All Its Glory - *Adam Tuliper* | **T02** WCF & Web API: Can We All Just Get Along?!? - *Miguel Castro* | **T03** Using Visual Studio Tools for Apache Cordova to Create Multi-Platform Applications - *Kevin Ford* | **T04** Developer Productivity in Visual Studio 2015 - *Robert Green* |
| 10:45 AM | 12:00 PM | **T05** Build Data-Driven Web Apps with ASP.NET Core - *Rachel Appel* | **T06** What's New in SQL Server 2016 - *Leonard Lobel* | **T07** From Oculus to HoloLens: Building Virtual & Mixed Reality Apps & Games - *Nick Landry* | **T08** This session is sequestered, details will be released soon |
| 12:00 PM | 1:30 PM | Lunch • Visit Exhibitors | | | |
| 1:30 PM | 2:30 PM | **GENERAL SESSION: JavaScript and the Rise of the New Virtual Machine**, *Scott Hanselman, Principal Community Architect for Web Platform and Tools, Microsoft* | | | |
| 2:45 PM | 4:00 PM | **T09** What You Need To Know About ASP.NET Core 1 and ASP.NET MVC - *Rachel Appel* | **T10** Database Development with SQL Server Data Tools - *Leonard Lobel* | **T11** This session is sequestered, details will be released soon | **T12** Linux and OSX for Windows Developers - *Brian Randell* |
| 4:15 PM | 5:30 PM | **T13** UX Beyond the Keyboard: Gaze, Speech and Other Interfaces - *John Alexander* | **T14** Cloud Enable .NET Client LOB Applications - *Robert Green* | **T15** Go Mobile with C#, Visual Studio, and Xamarin - *James Montemagno* | **T16** Busting .NET Myths - *Jason Bock* |
| 5:30 PM | 6:45 PM | Welcome Reception | | | |

| START TIME | END TIME | Visual Studio Live! Day 2: Wednesday, May 18, 2016 | | | |
|---|---|---|---|---|---|
| 8:00 AM | 9:15 AM | **W01** Angular 2 101 - *Deborah Kurata* | **W02** Building for the Internet of Things: Hardware, Sensors & the Cloud - *Nick Landry* | **W03** Creating Great Looking Android Applications Using Material Design - *Kevin Ford* | **W04** DevOps for Developers - *Brian Randell* |
| 9:30 AM | 10:45 AM | **W05** Hack Proofing Your Modern Web Applications - *Adam Tuliper* | **W06** Setting Up Your First VM, and Maybe Your Data Center, In Azure - *Eric D. Boyd* | **W07** Developing with Xamarin & Amazon AWS to Scale Native Cross-Platform Mobile Apps - *James Montemagno* | **W08** Easy and Fun Environment Creation with DevOps Provisioning Tools and Visual Studio - *John Alexander* |
| 11:00 AM | 12:00 PM | **GENERAL SESSION: Coding, Composition, and Combinatorics**, *Billy Hollis, Next Version Systems* | | | |
| 12:00 PM | 1:30 PM | Birds-of-a-Feather Lunch • Visit Exhibitors | | | |
| 1:30 PM | 2:45 PM | **W09** AngularJS & ASP.NET MVC Playing Nice - *Miguel Castro* | **W10** Knockout in 75 Minutes (Or Less...) - *Christopher Harrison* | **W11** No Schema, No Problem!—Introduction to Azure DocumentDB - *Leonard Lobel* | **W12** Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - *Benjamin Day* |
| 3:00 PM | 4:15 PM | **W13** Angular 2 Forms and Validation - *Deborah Kurata* | **W14** Introduction to the Next Generation of Azure PaaS—Service Fabric and Containers - *Vishwas Lele* | **W15** Big Data and Hadoop with Azure HDInsight - *Andrew Brust* | **W16** Effective Agile Software Requirements - *Richard Hundhausen* |
| 4:30 PM | 5:45 PM | **W17** Busy JavaScript Developer's Guide to ECMAScript 6 - *Ted Neward* | **W18** Building "Full Stack" Applications with Azure App Service - *Vishwas Lele* | **W19** Predicting the Future Using Azure Machine Learning - *Eric D. Boyd* | **W20** Acceptance Testing in Visual Studio 2015 - *Richard Hundhausen* |
| 7:00 PM | 9:00 PM | Rollin' On the River Bat Cruise | | | |

| START TIME | END TIME | Visual Studio Live! Day 3: Thursday, May 19, 2016 | | | |
|---|---|---|---|---|---|
| 8:00 AM | 9:15 AM | **TH01** Role-Based Security Stinks: How to Implement Better Authorization in ASP.NET & WebAPI - *Benjamin Day* | **TH02** Code Reactions—An Introduction to Reactive Extensions - *Jason Bock* | **TH03** Busy Developer's Guide to NoSQL - *Ted Neward* | **TH04** Open Source Software for Microsoft Developers - *Rockford Lhotka* |
| 9:30 AM | 10:45 AM | **TH05** Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - *Benjamin Day* | **TH06** DI Why? Getting a Grip on Dependency Injection - *Jeremy Clark* | **TH07** Power BI 2.0: Analytics in the Cloud and in Excel - *Andrew Brust* | **TH08** Architects? We Don't Need No Stinkin' Architects! - *Michael Stiefel* |
| 11:00 AM | 12:15 PM | **TH09** Future of the Web with HTTP2 - *Ben Dewey* | **TH10** Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark* | **TH11** User Experience Case Studies—Good and Bad - *Billy Hollis* | **TH12** This session is sequestered, details will be released soon |
| 12:15 PM | 1:30 PM | Lunch | | | |
| 1:30 PM | 2:45 PM | **TH13** TypeScript: Work Smarter Not Harder with JavaScript - *Allen Conway* | **TH14** Exceptional Development: Dealing With Exceptions in .NET - *Jason Bock* | **TH15** Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - *Billy Hollis* | **TH16** Clean Code: Homicidal Maniacs Read Code, Too! - *Jeremy Clark* |
| 3:00 PM | 4:15 PM | **TH17** Unit Testing JavaScript Code in Visual Studio .NET - *Allen Conway* | **TH18** Async Patterns for .NET Development - *Ben Dewey* | **TH19** Windows 10 Design Guideline Essentials - *Billy Hollis* | **TH20** Architecting For Failure: How to Build Cloud Applications - *Michael Stiefel* |

*Speakers and sessions subject to change*

**DETAILS COMING SOON! These sessions have been sequestered by our conference chairs. Be sure to check vslive.com/austin for session updates!**

**CONNECT WITH VISUAL STUDIO LIVE!**

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

**VSLIVE.COM/AUSTIN**

# Writing Windows Services in PowerShell

Jean-François Larvoire

Windows Services normally are compiled programs written in C, C++, C# or other Microsoft .NET Framework-based languages, and debugging such services can be fairly difficult. A few months ago, inspired by other OSes that allow writing services as simple shell scripts, I began to wonder if there could be an easier way to create them in Windows, as well.

This article presents the end result of that effort: A novel and easy way to create Windows Services, by writing them in the Windows PowerShell scripting language. No more compilation, just a quick edit/test cycle that can be done on any system, not just the developer's own.

I provide a generic service script template called PSService.ps1, which allows you to create and test new Windows Services in minutes, with just a text editor like Notepad. This technique can save a lot of time and development effort for anyone who wants to experiment with Windows Services—or even provide real services for Windows when performance isn't a critical factor. PSService.ps1 can be downloaded from bit.ly/1Y0XRQB.

This article discusses:

- Windows Services architecture
- Managing Windows Services
- Using C# code snippets in Windows PowerShell scripts
- Writing a self-managing service in Windows PowerShell

Technologies discussed:

Windows Services, Windows PowerShell, C#

Code download available at:

bit.ly/1Y0XRQB

## What Is a Windows Service?

Windows Services are programs that run in the background, with no user interaction. For example, a Web server, which silently responds to HTTP requests for Web pages from the network, is a service, as is a monitoring application that silently logs performance measurements or records hardware sensor events.

Services can start automatically when the system boots. Or they can start on demand, as requested by applications that rely on them. Services run in their own Windows session, distinct from the UI session. They run in a number of system processes, with carefully selected rights to limit security risks.

## The Windows Service Control Manager

The services are managed by the Windows Service Control Manager (SCM). The SCM is responsible for configuring services, starting them, stopping them and so forth.

The SCM control panel is accessible via Control Panel | System and Security | Administrative Tools | Services. As **Figure 1** shows, it displays a list of all configured services, with their name, description, status, startup type and user name.

There are also command-line interfaces to the SCM:

- The old net.exe tool, with its well-known "net start" and "net stop" commands, dates from as far back as MS-DOS! Despite its name, it can be used to start and stop any service, not just network services. Type "net help" for details.

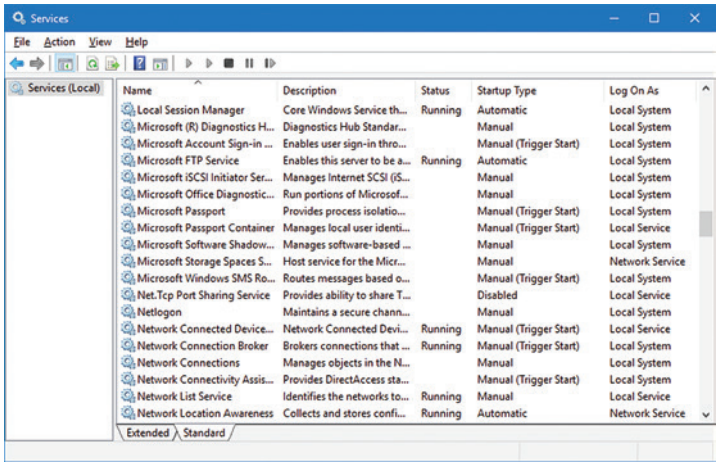



Figure 1 **The Windows Service Control Manager GUI in Windows 10**

- A more powerful tool called sc.exe, introduced in Windows NT, gives fine control over all aspects of service management. Type "sc /?" for details.

These command-line tools, although still present in Windows 10, are now deprecated in favor of Windows PowerShell service management functions, described later.

Gotcha: Both net.exe and sc.exe use the "short" one-word service name, which, unfortunately, isn't the same as the more descriptive name displayed by the SCM control panel. To get the correspondence between the two names, use the Windows PowerShell get-service command.

## Service States

Services can be in a variety of states. Some states are required, others are optional. The two basic states that all services must support are stopped and started. These show up respectively as (blank) or Running in under the Status column in **Figure 1**.

A third optional state is Paused. And another implicit state every service supports even if it's not mentioned is Uninstalled.

A service can make transitions between those states, as shown in **Figure 2**.

Finally, there are also several transitory states that services may optionally support: StartPending, StopPending, PausePending, ContinuePending. These are useful only if state transitions take a significant amount of time.

## Windows PowerShell Service Management Functions

Windows PowerShell has been the recommended system management shell since Windows Vista. It includes a powerful scripting language and a large library of functions for managing all aspects of the OS. Some of Windows PowerShell strengths are:

- Consistent function names
- Fully object-oriented
- Easy management of any .NET object

Windows PowerShell provides many service management functions, which are known as cmdlets. **Figure 3** shows some examples.

For a complete list of all commands with the string "service" in their names, run:

```
Get-Command *service*
```

For a list of just the service management functions, run:

```
Get-Command -module Microsoft.PowerShell.Management *service*
```

Surprisingly, there's no Windows PowerShell function for removing (that is, uninstalling) a service. This is one of the rare cases when it's still necessary to use the old sc.exe tool:

```
sc.exe delete $serviceName
```

## The .NET ServiceBase Class

All services must create a .NET object deriving from the ServiceBase class. Microsoft documentation describes all properties and methods of that class. **Figure 4** lists a few of these, of particular interest for this project.

By implementing these methods, a service application will be manageable by the SCM to start automatically at boot time or on demand; and it'll be manageable by the SCM control panel, by the old net.exe and sc.exe commands, or by the new Windows PowerShell service management functions, to start or stop manually.

> All services must create a .NET object deriving from the ServiceBase class.

## Creating an Executable from C# Source Embedded in a Windows PowerShell Script

PowerShell makes it easy to use .NET objects in a script. By default it has built-in support for many .NET object types, sufficient for most purposes. Better still, it's extensible and allows embedding short C# code snippets in a Windows PowerShell script to add support for any other .NET feature. This capability is provided by the Add-Type command, which, despite its name, can do much more than just adding support for new .NET object types to Windows PowerShell. It can even compile and link a complete C# application into a new executable. For example, this hello.ps1 Windows PowerShell script:

```
$source = @"
  using System;
  class Hello {
    static void Main() {
      Console.WriteLine("Hello World!");
    }
  }
"@
Add-Type -TypeDefinition $source -Language CSharp -OutputAssembly "hello.exe"
  -OutputType ConsoleApplication
```

will create a hello.exe application, that prints "Hello world!":

```
PS C:\Temp> .\hello.ps1
PS C:\Temp> .\hello.exe
Hello World!
PS C:\Temp>
```

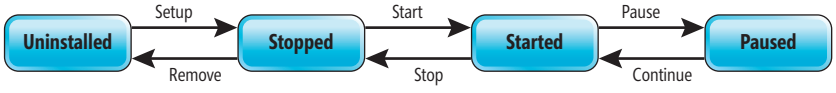



Figure 2 **Service States**

Figure 3 **Windows PowerShell Service Management Functions**

| Function Name | Description |
|---|---|
| Start-Service | Starts one or more stopped services |
| Stop-Service | Stops one or more running services |
| New-Service | Installs a new service |
| Get-Service | Gets the services on a local or remote computer, with their properties |
| Set-Service | Starts, stops and suspends a service, and changes its properties |

Figure 4 **Some Properties and Methods of the ServiceBase Class**

| Member | Description |
|---|---|
| ServiceName | Short name used to identify the service to the system |
| CanStop | Whether the service can be stopped once it has started |
| OnStart() | Actions to take when the service starts |
| OnStop() | Actions to take when the service stops |
| Run() | Registers the service executable with the SCM |

## Putting It All Together

**PSService.ps1 Features** Based on all I've discussed so far, I can now create that Windows PowerShell service I've been dreaming about, a PSService.ps1 script that can:

- Install and uninstall itself (using Windows PowerShell service management functions).
- Start and stop itself (using the same set of functions).
- Contain a short C# snippet, which creates the PSService.exe that the SCM expects (using the Add-Type command).
- Make the PSService.exe stub call back into the PSService.ps1 script for the actual service operation (in reponse to OnStart, OnStop and other events).
- Be manageable by the SCM control panel and all command-line tools (thanks to the PSService.exe stub).
- Be resilient, and process successfully any command when in any state. (For example, it can automatically stop the service before uninstalling it, or do nothing when asked to start an already started service.)
- Support Windows 7 and all later versions of Windows (using only Windows PowerShell v2 features).

Note that I'll cover only the critical parts of PSService.ps1 design and implementation in this article. The sample script also contains debugging code, and some support for optional service features, but their description would needlessly complicate the explanations here.

Figure 5 **Global Variable Defaults**

| Variable | Description | Default |
|---|---|---|
| $serviceName | A one-word name used for net start commands, and others | The base name of the script |
| $serviceDisplayName | A more descriptive name for the service | A Sample PowerShell Service |
| $installDir | Where to install the service files | ${ENV:windir}\System32 |
| $logFile | Name of the file in which to log the service messages | ${ENV:windir}\Logs\ $serviceName.log |
| $logName | Name of the Event Log in which to record service events | Application |

**PSService.ps1 Architecture** The script is organized in a series of sections:

- A header comment describing the file.
- A comment-based help block.
- The Param block defining command-line switches.
- Global variables.
- Helper routines: Now and Log.
- A C# source block of the PSService.exe stub.
- The main routine, processing every command-line switch.

## Global Settings

Immediately beneath the Param block, PSService.ps1 contains global variables defining global settings, which can be changed as needed. The defaults are shown in **Figure 5**.

Using the base name of the file as the service name (for example, PSService for PSService.ps1) lets you create multiple services from the same script, just by copying the script, renaming the copy, then installing the copy.

## Command-Line Arguments

To make it easy to use, the script supports command-line arguments that match all state transitions, as shown in **Figure 6**.

(Support for the paused state isn't implemented but would be easy to add, with the corresponding state transition options.)

**Figure 7** shows a few more management arguments that the script supports.

Each state transition switch has two modes of operation:

- When invoked by the end user: Use the Windows PowerShell service management functions to trigger a state transition.
- When invoked by the SCM (indirectly via the service.exe stub): Manage the service.ps1 service instance accordingly.

The two cases can be distinguished at run time by checking the user name: In the first case it's a normal user (the system administrator); in the second case it's the actual Windows system user. The system user can be identified like this:

```
$identity = [Security.Principal.WindowsIdentity]::GetCurrent()
$userName = $identity.Name   # Ex: "NT AUTHORITY\SYSTEM" or "Domain\Administrator"
$isSystem = ($userName -eq "NT AUTHORITY\SYSTEM")
```

## Installation

The goal of a service installation is to store a copy of the service files in a local directory, then to declare this to the SCM, so that it knows which program to run to start the service.

Here's the sequence of operations performed by the -Setup switch processing:

1. Uninstall any previous instance, if any.
2. Create the installation directory if needed. (This isn't needed for the default: C:\Windows\System32.)
3. Copy the service script into the installation directory.
4. Create a service.exe stub in that same installation directory, from the C# snippet in the script.
5. Register the service.


# 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

Open Source

NCache™

FREE Download

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

Figure 6 **Command-Line Arguments for State Transitions**

| Switch | Description |
|---|---|
| -Start | Start the service |
| -Stop | Stop the service |
| -Setup | Install itself as a service |
| -Remove | Uninstall the service |

Figure 7 **Supported Management Arguments**

| Switch | Description |
|---|---|
| -Restart | Stop the service, then start it again |
| -Status | Display the current state of the service |
| -Service | Run the service instance (for use only by the service.exe stub) |
| -Version | Display the service version |
| Common Parameters | -? , -Verbose , -Debug and so forth |

Note that starting with a single Windows PowerShell source script (PSService.ps1), I end up with three files installed in C:\Windows\System32: PSService.ps1, PSService.pdb and PSService.exe. These three files will need to be removed during uninstallation. The installation is implemented by including two pieces of code in the script:

1. The definition of the -Setup switch in the Param block at the beginning of the script:
```
[Parameter(ParameterSetName='Setup', Mandatory=$true)]
[Switch]$Setup,    # Install the service
```
2. An if block, as shown in **Figure 8,** for processing the -Setup switch in the main routine at the end of the script.

## Startup

The authority responsible for managing services is the SCM. Every startup operation must go through the SCM so it can keep track of service states. So even if the user wants to manually initiate a startup using the service script, that startup must be done through a request to the SCM. In this case, the sequence of operations is:

1. The user (an administrator) runs a first instance: PSService.ps1 -Start.
2. This first instance tells the SCM to start the service: Start-Service $serviceName.
3. The SCM runs PSService.exe. Its Main routine creates a service object, then invokes its Run method.
4. The SCM invokes the service object OnStart method.
5. The C# OnStart method runs a second script instance: PSService.ps1 -Start.
6. This second instance, now running in the background as the system user, starts a third instance, which will remain in memory as the actual service: PSService.ps1 -Service. It's this last -Service instance that does the actual service task, which you customize for whatever task is desired.

In the end, there will be two tasks running: PSService.exe, and a PowerShell.exe instance running PSService.ps1 -Service.

> The authority responsible for managing services is the SCM.

All this is implemented by having three pieces of code in the script:

1. The definition of the -Start switch in the Param block at the beginning of the script:
```
[Parameter(ParameterSetName='Start', Mandatory=$true)]
 [Switch]$Start, # Start the service
```
2. In the main routine, at the end of the script, an if block processing the -Start switch:
```
if ($Start) {# Start the service
  if ($isSystem) { # If running as SYSTEM, ie. invoked as a service
    Start-Process PowerShell.exe -ArgumentList (
      "-c & '$scriptFullName' -Service")
  } else { # Invoked manually by the administrator
  Start-Service $serviceName # Ask Service Control Manager to start it
  }
  return
}
```
3. In the C# source snippet, a Main routine and a handler for the OnStart method that runs PSService.ps1 -Start, as shown in **Figure 9**.

Figure 8 **Setup Code Handler**

```
if ($Setup) {                                # Install the service
  # Check if it's necessary (if not installed,
  # or if this script is newer than the installed copy).
  [...] # If necessary and already installed, uninstall the old copy.
  # Copy the service script into the installation directory.
  if ($ScriptFullName -ne $scriptCopy) {
    Copy-Item $ScriptFullName $scriptCopy
  }
  # Generate the service .EXE from the C# source embedded in this script.
  try {
    Add-Type -TypeDefinition $source -Language CSharp -OutputAssembly $exeFullName
      -OutputType ConsoleApplication -ReferencedAssemblies "System.ServiceProcess"
  } catch {
    $msg = $_.Exception.Message
    Write-error "Failed to create the $exeFullName service stub. $msg"
    exit 1
  }
  # Register the service
  $pss = New-Service $serviceName $exeFullName -DisplayName $serviceDisplayName
    -StartupType Automatic
  return
}
```

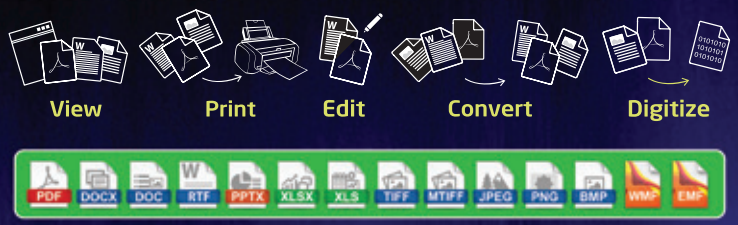
Figure 9 **Start Code Handler**

```
public static void Main() {
  System.ServiceProcess.ServiceBase.Run(new $serviceName());
}
protected override void OnStart(string [] args) {
  // Start a child process with another copy of this script.
  try {
    Process p = new Process();
    // Redirect the output stream of the child process.
    p.StartInfo.UseShellExecute = false;
    p.StartInfo.RedirectStandardOutput = true;
    p.StartInfo.FileName = "PowerShell.exe";
    p.StartInfo.Arguments = "-c & '$scriptCopyCname' -Start";
    p.Start();
    // Read the output stream first and then wait. (Supposed to avoid deadlocks.)
    string output = p.StandardOutput.ReadToEnd();
    // Wait for the completion of the script startup code,
    // which launches the -Service instance.
    p.WaitForExit();
  } catch (Exception e) {
    // Log the failure.
  }
}
```

www.gnostice.com

Gnostice™
Smart needs...Smarter solutions...

# One component suite for all Document requirements








Visual Studio MAGAZINE 2 READER'S CHOICE 2015 AWARD WINNER

WinForms, WPF, Web Forms, MVC, HTML5

Download your FREE 30-day trial today!!!

Gnostice XtremeDocumentStudio .NET enables you to develop rich and engaging user experiences. With XtremeDocumentStudio you can painlessly embed document viewing, printing, conversion, template-driven document creation, editing, mail-merge, digitization (OCR) and a host of other functionality in your applications. Format support includes PDF, DOCX, DOC, RTF, PPTX, XLSX, XLS, TIFF, MTIFF, JPEG, PNG, BMP, WMF & EMF. Platform support includes WinForms, WPF, Web Forms, MVC & HTML5. XtremeDocumentStudio .NET does not require external software or libraries such as Microsoft Word, Open XML SDK, Word Automation Services, Adobe PDF library or GhostScript.

XtremeDocumentStudio .NET

## Getting the Service State

The -Status handler simply asks the SCM for the service status, and sends it to the output pipe:

```
try {
  $pss = Get-Service $serviceName -ea stop # Will error-out if not installed.
} catch {
  "Not Installed"
  return
}
$pss.Status
```

But during the debugging phase, you might encounter script failures, due, for example, to syntax errors in the script and the like. In such cases, the SCM status might end up being incorrect. I've actually run into this several times while preparing this article. To help diagnose that kind of thing, it's prudent to double-check and search for -Service instances:

```
$spid = $null
$processes = @(gwmi Win32_Process -filter "Name = 'powershell.exe'" | where {
  $_.CommandLine -match ".*$scriptCopyCname.*-Service"
})
foreach ($process in $processes) { # Normally there is only one.
  $spid = $process.ProcessId
  Write-Verbose "$serviceName Process ID = $spid"
}
if (($pss.Status -eq "Running") -and (!$spid)) {
# This happened during the debugging phase.
  Write-Error "The Service Control Manager thinks $serviceName is started,
    but $serviceName.ps1 -Service is not running."
  exit 1
}
```

## Stop and Uninstallation

The Stop and Remove operations basically undo what Setup and Start did:

- -Stop (if invoked by the user) tells the SCM to stop the service.
- If invoked by the system, it simply kills the PSService.ps1 -Service instance.
- -Remove stops the service, unregisters it using sc.exe delete $serviceName, then deletes the files in the installation directory.

The implementation is also very similar to that of the Setup and Start:

1. The definition of each switch in the Param block at the beginning of the script.
2. An if block processing the switch in the main routine, at the end of the script.
3. For the stop operation, in the C# source snippet, a handler for the OnStop method that runs PSService.ps1 -Stop. The stop operation does things differently depending on whether the user is a real user or the system.

## Event Logging

Services run in the background, without a UI. This makes them difficult to debug: How can you diagnose what went wrong, when by design nothing is visible? The usual method is to keep a record of all error messages with time stamps, and also to log important events that went well, such as state transitions.

Services run in the background, without a UI. This makes them difficult to debug: How can you diagnose what went wrong, when by design nothing is visible?

The sample PSService.ps1 script implements two distinct logging methods, and uses both at strategic points (including in parts of the previous code extracts, removed here to clarify the basic operation):

- It writes event objects into the Application log, with the service name as the source name, as shown in **Figure 10**. These event objects are visible in the Event Viewer, and can be filtered and searched using all the capabilities of that tool. You can also get these entries with the Get-Eventlog cmdlet:

```
Get-Eventlog -LogName Application -Source PSService | select -First 10
```

- It writes message lines to a text file in the Windows Logs directory, ${ENV:windir}\Logs\$serviceName.log, as shown in **Figure 11**. This log file is readable with Notepad, and can be searched using findstr.exe, or Win32 ports of grep, tail and so forth.

A Log function makes it easy to write such messages, automatically prefixing the ISO 8601 time stamp and current user name:

```
Function Log ([String]$string) {
  if (!(Test-Path $logDir)) {
    mkdir $logDir
  }
  "$(Now) $userName $string" |
    out-file -Encoding ASCII -append "$logDir\$serviceName.log"
}
```

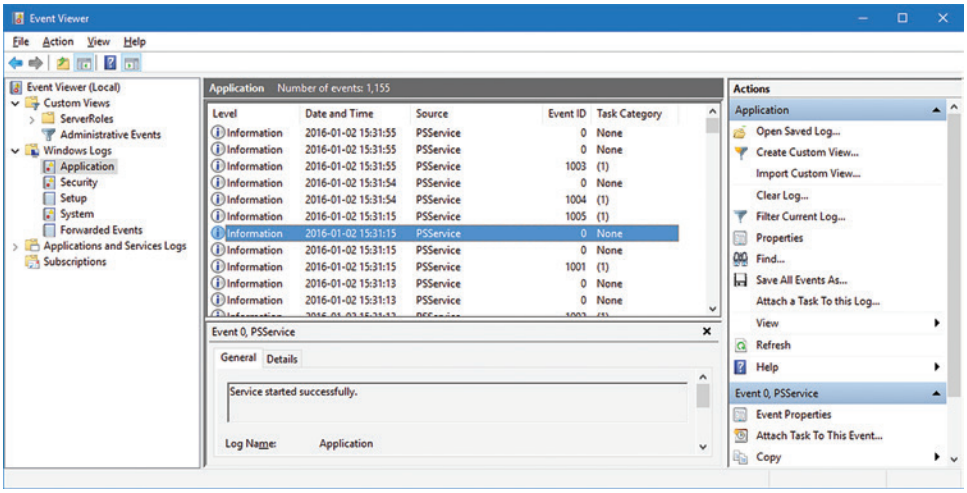



Figure 10 **Event Viewer with PSService Events**

## Sample Test Session

Here's how the preceding logs were generated:

```
PS C:\Temp> C:\SRC\PowerShell\SRC\PSService.ps1 -Status
Not Installed
PS C:\Temp> PSService.ps1 -Status
PSService.ps1 : The term 'PSService.ps1' is not recognized as the name of
a cmdlet, function, script file, or operable program.
[...]
PS C:\Temp> C:\SRC\PowerShell\SRC\PSService.ps1 -Setup
PS C:\Temp> PSService.ps1 -Status
Stopped
PS C:\Temp> PSService.ps1 -Start
PS C:\Temp>
```

This shows how to use the service in general. Keep in mind that it must be run by a user with local admin rights, in a Windows PowerShell session running as Administrator. Notice how the PSService.ps1 script wasn't on the path at first, then after the -Setup operation it is. (The first -Status call with no path specified fails; the second -Status call succeeds.)

> A service script written in Windows PowerShell will be very good for prototyping a concept.

Calling PSService.ps1 -Status at this stage would produce this output: Running. And this, after waiting 30 seconds:

```
PS C:\Temp> PSService.ps1 -Stop
PS C:\Temp> PSService.ps1 -Remove
PS C:\Temp>
```

## Customizing the Service

To create your own service, just do the following:

- Copy the sample service into a new file with a new base name, such as C:\Temp\MyService.ps1.
- Change the long service name in the global variables section.
- Change the TO DO block in the -Service handler at the end of the script. Currently, the while ($true) block just contains dummy code that wakes up every 10 seconds and logs one message in the log file:

```
######### TO DO: Implement your own service code here. ##########
###### Example that wakes up and logs a line every 10 sec: ######
Start-Sleep 10
Log "$script -Service # Awaken after 10s"
```

- Install and start testing:

```
C:\Temp\MyService.ps1 -Setup
MyService.ps1 -Start
type C:\Windows\Logs\MyService.log
```

You shouldn't have to change anything in the rest of the script, except to add support for new SCM features like the Paused state.

## References


Introduction to Windows Service Applications (bit.ly/1UOBJJY)

How to: Create Windows Services (bit.ly/1VJCnJo)

ServiceBase Class (bit.ly/1UOC13y)

Managing Services (bit.ly/1VJCZyG)

How to: Debug Windows Service Applications (bit.ly/1RjEhPg)


Figure 11 **Sample Log File**

```
PS C:\Temp> type C:\Windows\Logs\PSService.log
2016-01-02 15:29:47 JFLZB\Larvoire C:\SRC\PowerShell\SRC\PSService.ps1 -Status
2016-01-02 15:30:38 JFLZB\Larvoire C:\SRC\PowerShell\SRC\PSService.ps1 -Setup
2016-01-02 15:30:42 JFLZB\Larvoire PSService.ps1 -Status
2016-01-02 15:31:13 JFLZB\Larvoire PSService.ps1 -Start
2016-01-02 15:31:15 NT AUTHORITY\SYSTEM & 'C:\WINDOWS\System32\PSService.
ps1' -Start
2016-01-02 15:31:15 NT AUTHORITY\SYSTEM PSService.ps1 -Start: Starting
script 'C:\WINDOWS\System32\PSService.ps1' -Service
2016-01-02 15:31:15 NT AUTHORITY\SYSTEM & 'C:\WINDOWS\System32\PSService.
ps1' -Service
2016-01-02 15:31:15 NT AUTHORITY\SYSTEM PSService.ps1 -Service #
Beginning background job
2016-01-02 15:31:25 NT AUTHORITY\SYSTEM PSService -Service # Awaken after 10s
2016-01-02 15:31:36 NT AUTHORITY\SYSTEM PSService -Service # Awaken after 10s
2016-01-02 15:31:46 NT AUTHORITY\SYSTEM PSService -Service # Awaken after 10s
2016-01-02 15:31:54 JFLZB\Larvoire PSService.ps1 -Stop
2016-01-02 15:31:55 NT AUTHORITY\SYSTEM & 'C:\WINDOWS\System32\PSService.
ps1' -Stop
2016-01-02 15:31:55 NT AUTHORITY\SYSTEM PSService.ps1 -Stop: Stopping
script PSService.ps1 -Service
2016-01-02 15:31:55 NT AUTHORITY\SYSTEM Stopping PID 34164
2016-01-02 15:32:01 JFLZB\Larvoire PSService.ps1 -Remove
PS C:\Temp>
```

## Limitations and Issues

The service script must be run in a shell running with administrator rights or you'll get various access denied errors.

The sample script works in Windows versions XP to 10, and the corresponding server versions. In Windows XP, you have to install Windows PowerShell v2, which isn't available by default. Download and install Windows Management Framework v2 for XP (bit.ly/1MpOdpV), which includes Windows PowerShell v2. Note that I've done very little testing in that OS, as it's not supported anymore.

On many systems, the Windows PowerShell script execution is disabled by default. If you get an error like, "the execution of scripts is disabled on this system," when trying to run PSService.ps1, then use:

```
Set-ExecutionPolicy RemoteSigned
```

For more information, see "References."

Obviously, a service script like this can't be as performant as a compiled program. A service script written in Windows PowerShell will be good for prototyping a concept, and for tasks with low performance costs like system monitoring, service clustering and so forth. But for any high performance task, a rewrite in C++ or C# is recommended.

The memory footprint is also not as good as that of a compiled program, because it requires loading a full-fledged Windows PowerShell interpreter in the System session. In today's world, with systems having many gigabytes of RAM, this is not a big deal.

This script is completely unrelated to Mark Russinovich's PsService.exe. I chose the PSService.ps1 name before I knew about the homonymy. I'll keep it for this sample script as I think the name makes its purpose clear. Of course, if you plan to experiment with your own Windows PowerShell service, you must rename it, to get a unique service name from a unique script base name! ■


**Jean-François Larvoire** *works for Hewlett-Packard Enterprise in Grenoble, France. He has been developing software for 30 years for PC BIOS, Windows drivers, Windows and Linux system management. He can be reached at jf.larvoire@hpe.com.*



**Thanks** to the following technical expert for reviewing this article: *Jeffery Hicks (JDH IT Solutions)*

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

VSLIVE.COM/BOSTON

# BETTER CODE FOR ALL

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

Boston

CAMBRIDGE, MA
JUNE 13-16, 2016
THE HYATT REGENCY

For the first time in a decade, Boston will host **VISUAL STUDIO LIVE!** from June 13 – 16. Through four intense days of practical, unbiased, Developer training, join us as we dig in to the latest features of Visual Studio 2015, ASP.NET, JavaScript, TypeScript, Windows 10 and so much more. Code with industry experts, get practical answers to your current challenges, and immerse yourself in what's to come on the .NET horizon.

## Life, liberty, and the pursuit of better code: register to join us today!

**REGISTER BY MAY 18 & SAVE $200!**

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

**USE PROMO CODE VSLMAY5**

PLATINUM SPONSOR
Microsoft Virtual Academy
LEARNING HAPPENS HERE

SILVER SPONSOR
Atalasoft from Lexmark

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

# BOSTON AGENDA AT-A-GLANCE

| ALM / DevOps | Cloud Computing | Database and Analytics | Mobile Client | Software Practices | UX / Design | Visual Studio / .NET Framework | Web Client | Web Server |
|---|---|---|---|---|---|---|---|---|

| START TIME | END TIME | Visual Studio Live! Pre-Conference Workshops: Monday, June 13, 2016 *(Separate entry fee required)* | | |
|---|---|---|---|---|
| 7:30 AM | 9:00 AM | Pre-Conference Workshop Registration • Coffee and Morning Pastries | | |
| 9:00 AM | 6:00 PM | **M01** Workshop: SQL Server for Developers - *Andrew Brust and Leonard Lobel* | **M02** Workshop: DevOps for Your Mobile Apps and Services - *Brian Randell* | **M03** Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries and Roy Cornelissen* |
| 6:45 PM | 9:00 PM | Dine-A-Round | | |

| START TIME | END TIME | Visual Studio Live! Day 1: Tuesday, June 14, 2016 | | | |
|---|---|---|---|---|---|
| 8:00 AM | 9:00 AM | Keynote: To Be Announced | | | |
| 9:15 AM | 10:30 AM | **T01** Technical Debt —Fight It with Science and Rigor - *Brian Randell* | **T02** What's New in SQL Server 2016 - *Leonard Lobel* | **T03** Getting Started with Aurelia - *Brian Noyes* | **T04** Developer Productivity in Visual Studio 2015 - *Robert Green* |
| 10:45 AM | 12:00 PM | **T05** Automated X-Browser Testing of Your Web Apps with Visual Studio CodedUI - *Marcel de Vries* | **T06** Database Lifecycle Management and the SQL Server Database - *Brian Randell* | **T07** Angular 2 101 - *Deborah Kurata* | **T08** This session is sequestered, details will be released soon |
| 12:00 PM | 1:30 PM | Lunch • Visit Exhibitors | | | |
| 1:30 PM | 2:45 PM | **T09** ASP.NET Core 1.0 in All Its Glory - *Adam Tuliper* | **T10** Predicting the Future Using Azure Machine Learning - *Eric D. Boyd* | **T11** TypeScript for C# Developers - *Chris Klug* | **T12** This session is sequestered, details will be released soon |
| 3:00 PM | 4:15 PM | **T13** Hack Proofing Your Modern Web Applications - *Adam Tuliper* | **T14** No Schema, No Problem!—Introduction to Azure DocumentDB - *Leonard Lobel* | **T15** Angular 2 Forms and Validation - *Deborah Kurata* | **T16** Windows 10—The Universal Application: One App To Rule Them All? - *Laurent Bugnion* |
| 4:15 PM | 5:30 PM | Welcome Reception | | | |

| START TIME | END TIME | Visual Studio Live! Day 2: Wednesday, June 15, 2016 | | | |
|---|---|---|---|---|---|
| 8:00 AM | 9:15 AM | **W01** AngularJS & ASP.NET MVC Playing Nice - *Miguel Castro* | **W02** Cloud Enable .NET Client LOB Applications - *Robert Green* | **W03** Creating Great Windows Universal User Experiences - *Danny Warren* | **W04** Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - *Nick Landry* |
| 9:30 AM | 10:45 AM | **W05** Did a Dictionary and a Func Just Become the New Black in ASP.NET Development? - *Chris Klug* | **W06** Azure Mobile Apps: APIs in the Cloud for Your Mobile Needs - *Danny Warren* | **W07** User Experience Case Studies—The Good and The Bad - *Billy Hollis* | **W08** Build Cross-Platform Mobile Apps with Ionic, Angular, and Cordova - *Brian Noyes* |
| 11:00 AM | 12:00 PM | **General Session:** More Personal Computing through Emerging Experiences, *Tim Huckaby, Founder / Chairman - InterKnowlogy & Actus Interactive Software* | | | |
| 12:00 PM | 1:30 PM | Birds-of-a-Feather Lunch • Visit Exhibitors | | | |
| 1:30 PM | 2:45 PM | **W09** Richer MVC Sites with Knockout JS - *Miguel Castro* | **W10** Breaking Down Walls with Modern Identity - *Eric D. Boyd* | **W11** Creating Dynamic Pages Using MVVM and Knockout.JS - *Christopher Harrison* | **W12** Mobile App Development with Xamarin and F# - *Rachel Reese* |
| 3:00 PM | 4:15 PM | **W13** Securing Client JavaScript Apps - *Brian Noyes* | **W14** Exploring Microservices in a Microsoft Landscape - *Marcel de Vries* | **W15** Learning to Live Without Data Grids in Windows 10 - *Billy Hollis* | **W16** Conquer the Network—Making Your C# Mobile App More Resilient to Network Hiccups - *Roy Cornelissen* |
| 4:30 PM | 5:45 PM | **W17** Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - *Benjamin Day* | **W18** Patterns and Practices for Real-World Event-Driven Microservices - *Rachel Reese* | **W19** Take Your Site From Ugh to OOH with Bootstrap - *Philip Japikse* | **W20** Strike Up a Conversation with Cortana on Windows 10 - *Walt Ritscher* |
| 6:15 PM | 8:30 PM | VSLive!'s Boston By Land and Sea Tour | | | |

| START TIME | END TIME | Visual Studio Live! Day 3: Thursday, June 16, 2016 | | | |
|---|---|---|---|---|---|
| 8:00 AM | 9:15 AM | **TH01** Windows Presentation Foundation (WPF) 4.6 - *Laurent Bugnion* | **TH02** Open Source Software for Microsoft Developers - *Rockford Lhotka* | **TH03** Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - *Benjamin Day* | **TH04** Windows for Makers: Raspberry Pi, Arduino & IoT - *Nick Landry* |
| 9:30 AM | 10:45 AM | **TH05** Power BI 2.0: Analytics in the Cloud and in Excel - *Andrew Brust* | **TH06** Dependencies Demystified - *Jason Bock* | **TH07** Automate Your Builds with Visual Studio Team Services or Team Foundation Server - *Tiago Pascoal* | **TH08** Automated UI Testing for iOS and Android Mobile Apps - *James Montemagno* |
| 11:00 AM | 12:15 PM | **TH09** Big Data and Hadoop with Azure HDInsight - *Andrew Brust* | **TH10** Improving Performance in .NET Applications - *Jason Bock* | **TH11** Cross Platform Continuous Delivery with Team Build and Release Management - Tiago Pascoal | **TH12** This session is sequestered, details will be released soon |
| 12:15 PM | 1:30 PM | Lunch | | | |
| 1:30 PM | 2:45 PM | **TH13** Top 10 Entity Framework Features Every Developer Should Know - *Philip Japikse* | **TH14** Architecting For Failure: How to Build Cloud Applications - *Michael Stiefel* | **TH15** JavaScript Patterns for the C# Developer - *Ben Hoelting* | **TH16** Developing with Xamarin & Amazon AWS to Scale Native Cross-Platform Mobile Apps - *James Montemagno* |
| 3:00 PM | 4:15 PM | **TH17** Pretty, Yet Powerful. How Data Visualization Transforms the Way We Comprehend Information - *Walt Ritscher* | **TH18** Architects? We Don't Need No Stinkin' Architects! - *Michael Stiefel* | **TH19** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day* | **TH20** Advanced Mobile App Development for the Web Developer - *Ben Hoelting* |

*Speakers and sessions subject to change*

**DETAILS COMING SOON! These sessions have been sequestered by our conference chairs. Be sure to check vslive.com/boston for session updates!**

# Nurturing Lean UX Practices

Karl Melder

In Visual Studio 2015, Microsoft delivered several new debugging and diagnostic features that are detailed by Andrew Hall in the March 2016 *MSDN Magazine* article, "Debugging Improvements in Visual Studio 2015" (msdn.com/magazine/mt683794). For the features that involve substantial changes to the UX, Microsoft adopted a "Lean UX" approach using iterative experiments with direct user feedback to inform the design.

I want to share with you the process that was used to design one of those features, PerfTips, along with the best practices, tips and tricks picked up along the way. The goal is to inspire and enable you and your teams to effectively fold customer feedback directly into your development processes.

## Lean UX

Lean UX is a complement to lean development practices that are trending in our industry. Eric Ries defined lean as the practice of "Build, Measure and Learn" in his 2011 book, "The Lean Startup" (Crown Business), where he describes a "business-hypothesis-driven experimentation" approach. Similarly, Lean UX is a set of principles and processes that focuses on very early and ongoing customer validation, where you conduct experiments to validate your user and product design hypothesis in extremely short cycles. Design iterations are done quickly with a focus on solving real user problems. A good reference is Jeff Gothelf's 2013 book, "Lean UX" (O'Reilly Media), where he provides guidance and worksheets to help teams bring clarity to what they believe or hope to achieve.

For the team delivering the debugging experience in Visual Studio, Lean UX is a highly collaborative approach where the entire team, including program managers, UX researchers, developers and UX designers, were involved in generating ideas, generating hypotheses, and interpreting what was heard and seen from our customers.

This article is about fully embracing customer feedback in the product development process. It's about failing forward faster. It's about getting feedback on your ideas without any working bits. It's about not just one team in the developer tools division doing this, but lots of teams fundamentally changing how features are designed in a lean development process.

## The Design Challenge

Microsoft technology has a rich source of data that can provide developers with an expedient way to diagnose issues. However, in the UX labs, users would repeatedly fall back to manually walking the execution of code despite the advantage that tools such as the Profiler provided. The instrumentation data bore out the low

This article discusses:

- Lean Development
- Lean UX
- Customer Feedback
- Product Design
- Feature Design

Technologies discussed:

Visual Studio, Diagnostics, Debugging, PerfTips, Conditional BreakPoints

usage of the Visual Studio Profiler despite the conviction that it can make finding performance issues a far more efficient process. For a tool such as Visual Studio, where a user spends eight or more hours a day working, asking a user to change his workstyle can be a tricky business. So, the team wanted to leverage the user's natural workstyle when debugging performance issues and deliver an ambient experience.

> Microsoft technology has a rich source of data that could provide users with an expedient way to diagnose issues.

Had a more traditional waterfall approach been taken, focus groups might have been conducted to get some early feedback, a detailed spec written and when coding was near complete, usability studies would have been scheduled. A user would have been given tasks that exercised the new features and triaged the issues found like is done with bugs. For Visual Studio 2015, a very different approach was taken.

## The Research Process

Instead of scheduling usability studies when working bits were available, two users were prescheduled every Friday for the majority of the product cycle. These days were informally referred to as "Quick Pulse Fridays." Users came in for about two hours, where their time was typically split across two to four experiments. For each experiment, a best guess was made as to how much time should be dedicated. Each experiment was about either helping Microsoft learn more about its users and how they work, or about trying out an idea. Design ideas had to survive at least three weeks of getting positive results in order to move forward with them. A positive result meant users either felt strongly it had value for them, increased discoverability, made it easier to use or could demonstrate improvements to key scenarios.

UX research is often categorized into quantitative and qualitative, where a combination of instrumentation/analytics and customer feedback guides business and product development. In the early qualitative research, feedback meant getting the users' reaction to ideas. The team took into account not only what they said, but their physical reaction, facial expressions and tone of voice. Users were also given a real task, like fixing a performance bug in an application without any assistance from the research team as they were observed, as shown in the photo in **Figure 1**. That meant letting the users struggle. The team would take video of them for future review and take notes on both what was heard and seen. Watching the users helped the team to understand their workstyle and identify unarticulated needs a user might not know to ask for, but could provide a dramatic improvement to the product.

What was critical to the team's success was not spending any time trying to convince customers to like an idea. The users were simply showed the idea in terms of what it would be like using it. Then the team stepped back and just listened, watched and asked questions that helped the team understand the users' viewpoints. The key to the team's success was the ability to detach itself from the idea or design that it might have felt strongly about.

Every week different participants were recruited for a steady flow of new perspectives. There was both an internal team and a vendor who recruited, screened and scheduled users. The team did not look for users who had specific expertise with diagnostics; rather, the recruiting profile was simply active users of Visual Studio. This meant that each week there were users with different skills, experiences and work contexts. This gave the team the opportunity to learn something new each week and let it identify the consistencies. The team could also evolve its ideas to succeed with a wider audience.

Equally important was balancing how the team interacted with the users. How a question was asked could dramatically affect the outcome and bias the conversation. The team developed the habit of always asking open-ended questions—where the probing questions were derived from what the user said or did. For example, if a user told the team they didn't like something, they were simply asked, "Tell us more about that." The team tried not to assume anything and challenged its assumptions and hypotheses at every opportunity. These skills are basic to the UX field and were adopted by everyone on the team. If you want to learn more about these interviewing techniques, I recommend Cindy Alvarez's 2014 book, "Lean Customer Development" (O'Reilly Media).

## Early Quick-Pulse Sessions and the Unshakeable Workstyle

Early in the product cycle, the team started with an idea for helping users monitor performance of their code. The team created a mockup and got it in front of the Quick Pulse Friday users. What was consistently heard, even after three weeks of design alterations, was that they weren't sure what it was for and that they "would probably turn it off!" It wasn't necessarily what the team wanted to hear, but it needed to hear.

However, while also watching users diagnose application issues, it became clear that the team needed a UX that was more directly part of the code navigation experience. Even though there were several debugger windows that provide additional information, it was difficult for users to pay attention to several windows at a time. The team observed many users keeping their focus in the code,

Figure 1 **A Research Session with a User**

often mentally "code walking" the execution. This may seem obvious to any developer reading this article, but what was fascinating was how unshakable that workstyle is despite the availability of additional tools that are meant to make that task more efficient.

The team started out envisioning ideas using Photoshop, where it would take an extremely experienced designer upward of a day to generate a mockup that could be used for feedback. Photoshop tends to lend itself to creating a UI with high fidelity. Instead, the team started using Microsoft PowerPoint and a storyboard add-in (aka.ms/jz35cp) that let everyone on the team quickly create medium-fidelity representations of their ideas. These storyboards gave users a sense of what it might look like, but they were rough enough for users to tell it was an in-progress design and that their input had direct impact. The net effect is that it was much easier to throw away a 30-minute investment when an experiment failed. Also, ideas could be tested that the team knew wouldn't work in practice, but the feedback from users would help generate new ideas.

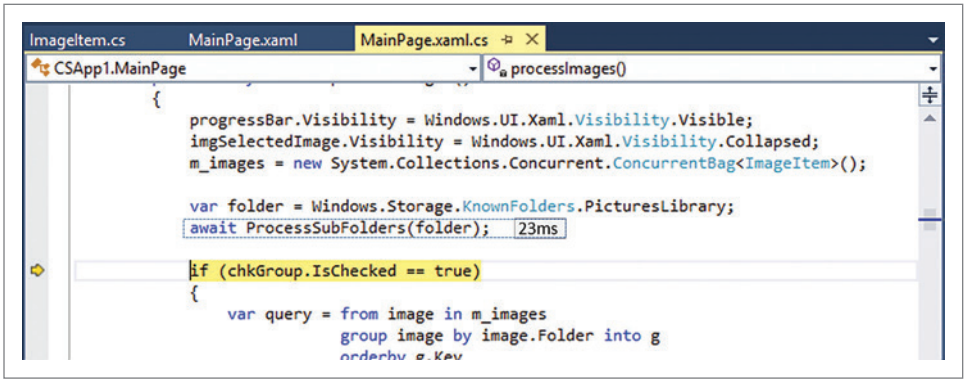



Figure 2 **An Early Mockup Showing Performance Data in a Debugging Session**

To get feedback on the user interaction model, each slide in the PowerPoint decks represented either a user action or a system response to that action. When drafting the interaction, a cursor icon image to show where the user would click would be included. This was useful when sharing ideas and working out the details. However, the cursor icon would be removed before showing it to the users. This allowed the team to ask the users what they would do next, providing a discount way of identifying possible discoverability issues. For each system-response slide the team would also ask if the users felt they were making progress, which let the team know if it was providing adequate feedback. This feedback technique is called a "cognitive walkthrough process" in UX research and can help you identify some issues at the very earliest stages of designing your interaction, while giving you an early sense of areas of concern that will require further iteration and experimentation to get right.

To gauge the potential impact of an idea, the team relied on the user's ability to articulate specifically how he might use the idea in his day-to-day work environment and what he perceived might be the direct benefits and drawbacks. The user had to provide detailed and plausible examples for the team to become confident the idea was worth pursuing. The team also looked to see if the user started to pay extra attention, get more animated and express excitement. The team was looking for ideas that would excite users and potentially have a very positive impact on their diagnostic experience.

## "Wow, This Is Amazing!"

The team needed a way to show performance information in the code that would not affect code readability and would give users an ambient in-code debugging experience. Code Lens, a feature in Visual Studio that lets you see information about the edit history, bugs, unit testing and references, provided inspiration about a potential interaction model and visual design. The team experimented with mockups of several ideas showing customers how, as a developer steps through code, it would show performance numbers in milliseconds (see **Figure 2**).

The earliest indication that the team was on to something was when a participant, who was a development manager, got very excited when he was walked through the experience. I should emphasize he was just showed the proposed experience without any background information. As he realized what he was seeing, he started asking detailed questions and got quite animated as he spoke. He said it would be a solution to a problem he was having with his novice developers making poor coding decisions that resulted in poor application performance. In his current process, performance issues were resolved through a labor-intensive code review process, which was a heavy tax on him and his team. He felt this idea could help his novice developers learn how to write performant code while they were first crafting their code. He made comments such as, "Can this [PerfTip] be policy [in Visual Studio]?" Another user, after recognizing its value, remarked, "What makes Visual Studio remarkable is the capabilities when you're on a line of code!"

This early feedback also got the team excited about this potential feature being an entry point for the diagnostic tools, solving some discoverability issues. The team hypothesized that these PerfTips could be a trigger for the users to venture into our richer tool set.

## Designing the Details

Everything done up to this point involved only mockups—with no investment in coding. If ideas got traction, greater levels of details were created in the PowerPoint "click-thru," as well as lots of design alternatives to experiment with weekly. However, the limit of what could be done with mockups was reached when several research issues remained:

- Validation that that the design for PerfTips when debugging common logic issues wasn't a distraction, but remained discoverable when dealing with performance issues.
- The team wanted the users to correctly interpret the performance numbers, which were timed from the last break in execution.
- Users had suggested only showing the values when performance was worrisome, but no one could confidently suggest a default threshold.

- There was concern the overhead of the debugger, which could add several milliseconds, might diminish its value to customers.

The team implemented a very minimal version of the feature that worked under specific conditions. Then an application with performance and logic issues for users to diagnose was created. Users were asked to identify the specific cause of the problem. If they weren't successful, the team could determine why the users weren't successful by what was heard and seen. The design could then be altered and tried again the following week. Also, during this time an external CTP version was delivered that was instrumented, where the PerfTip was linked to the properties window so users could easily change the threshold if they wanted. The team concluded:

- PerfTips were not a distraction when users were fixing logic issues. In fact, PerfTips needed to be tweaked with subtle animation to make them more noticeable when users were dealing with performance issues.
- Some simple phrasing changes, like adding the word "elapsed," cleared up any confusion users had about interpreting the timing data.
- Thresholds only confused users when they didn't show up consistently and a simple value that would work in most circumstances could not be identified. Some users said because they knew their code best, they would be the best judge of what would be reasonable performance times.
- Users recognized that the values would not be exact because of debugger overhead, but they said repeatedly they were fine with it as they would be looking at gross differences.

Overall, over the several weeks of iterations, the team got consistently positive results when tasking users to identify the source of performance issues.

Overall, over the several weeks of iterations, the team got consistently positive results when tasking users to identify the source of performance issues. Without any prompting, users also gave enthusiastic feedback with comments such as, "Fantastic," and, "Wow, this is amazing!"

## Taking Notes

When taking notes, the team learned to avoid drawing any conclusions until after the session when there was time to sit down together to discuss what happened. What was more useful was to take very raw notes in real time, trying to write down verbatim everything users said and what they did. Grammar and spelling was of no concern. These notes became the team's reference when refreshing itself on what happened and let it draw insights from the patterns that were seen over several weeks.

Microsoft OneNote became a very handy tool to track what the team was planning to test, capture raw notes and draft quick summaries. There was always a designated note-taker who captured what was heard and seen. This gave the other team members breathing room to completely focus on the user. For those who could not attend, the live sessions were shared with the team using Skype; everyone on the team was invited to watch and learn. Also, the sessions were recorded for team members who had meeting conflicts and wanted to watch later. The video recordings also let the team review areas that needed extra attention. The team discussion about the results each week informed what would be done the following week, where writing a formal report was unnecessary and would have just slowed everything down.

## Wrapping Up

The design and development of PerfTips was only a slice of what was done in the weekly experiments. Many ideas were explored, with as many as four experiments per user each week. The Breakpoint Settings redesign is another example of the experiments that were run week to week to iterate toward providing a more useful and usable experience. By applying Lean UX the team was able to mitigate risk, while finding inspiration from what was heard and seen during the experiments. These experiments took the guesswork out of the equation when designing the features. Ideas came from many sources and were inspired by watching how developers naturally worked.

If users couldn't see the value in an idea, the low cost to create a mockup made it easy to start over. Also, failures sparked new ideas. I hope the examples and tips for Lean UX will inspire you to give it a try. The "Lean" series of books referenced in this article will serve you well as a guide and a framework for adopting this approach.

## Participate in the Program

The Microsoft UX research team is looking for all types of developers to give direct feedback, as well as participate in this ongoing experiment. To sign up, include a few things about your technical background and how to best contact you at aka.ms/VSUxResearch.

I wish to give special thanks to all the folks who were involved in one way or another with this project. You can only describe the Quick Pulse Fridays as "crowded," with the team watching, learning and thinking very hard about delivering a well-thought-out and purposeful addition to Visual Studio. Special thanks need to go to Dan Taylor who had to stay ahead of the development team and who navigated the technological challenges with aplomb. Andrew Hall kept the team moving forward with his deep technical knowledge and pragmatic approach. Frank Wu kept the design ideas coming and had an uncanny ability to boil down an idea and find a way to keep it simple. ■

**Karl Melder** *is a senior UX researcher who has been steadily applying his education and experience in UX research, computer science, UI and human factors to design UXes. For the past 15-plus years he's been working to enhance the development experience in Visual Studio for a wide variety of customers.*

**Thanks** to the following Microsoft technical experts for reviewing this article: *Andrew Hall, Dan Taylor and Frank Wu*

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

VSLIVE.COM/REDMOND

# GIVE YOUR CODE A VOICE

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

## Microsoft HQ

MICROSOFT HQ
AUGUST 8-12, 2016
REDMOND, WA

**VISUAL STUDIO LIVE!** is on a Campaign for Code in 2016, and we're taking a swing through our home state, rallying @ Microsoft Headquarters in beautiful Redmond, WA. From August 8 – 12, developers, software architects, engineers, designers and more will convene for five days of unbiased and cutting-edge education on the Microsoft Platform. Plus, you'll be at the heart of it all: eating lunch with the Blue Badges, rubbing elbows with Microsoft insiders, exploring the campus, all while expanding your development skills and the ability to create better apps!

## Have your code heard: register to join us today!

**REGISTER NOW & SAVE $400!**

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

**USE PROMO CODE VSLRED2**

EVENT PARTNER
Microsoft

PLATINUM SPONSOR
LEARNING HAPPENS HERE

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH, OUR BUSINESS.

# REDMOND AGENDA AT-A-GLANCE

**DETAILS COMING SOON!** With all of the announcements coming out of Build, we'll be finalizing the Redmond Keynotes and the FULL Track of Microsoft-led sessions shortly. Be sure to check vslive.com/redmond for session updates!

| ALM / DevOps | Cloud Computing | Database & Analytics | Mobile Client | Software Practices | Visual Studio / .NET Framework | Web Client | Web Server | Windows Client | Microsoft Sessions |
|---|---|---|---|---|---|---|---|---|---|

| START TIME | END TIME | | | | | |
|---|---|---|---|---|---|---|
| **Visual Studio Live! Pre-Conference Workshops: Monday, August 8, 2016** *(Separate entry fee required)* | | | | | | |
| 8:00 AM | 5:30 PM | **M01** Workshop: DevOps for Your Mobile Apps and Services - *Brian Randell* | **M02** Workshop: Developer Dive into SQL Server 2016 - *Leonard Lobel* | **M03** Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka & Jason Bock* | | |
| 12:00 PM | 2:00 PM | Lunch @ The Mixer • Visit the Microsoft Company Store & Visitor Center | | | | |
| 7:00 PM | 9:00 PM | Dine-A-Round Dinner | | | | |
| **START TIME** | **END TIME** | **Visual Studio Live! Day 1: Tuesday, August 9, 2016** | | | | |
| 8:30 AM | 9:30 AM | **KEYNOTE**: To Be Announced, ***Amanda Silver***, *Director of Program Management, Visual Studio, Microsoft* | | | | |
| 9:45 AM | 11:00 AM | **T01** Windows Presentation Foundation (WPF) 4.6 - *Laurent Bugnion* | **T02** Angular 2 101 - *Deborah Kurata* | **T03** Go Mobile with C#, Visual Studio, and Xamarin - *James Montemagno* | **T04** What's New in SQL Server 2016 - *Leonard Lobel* | **T05** This Microsoft session is sequestered, details will be released soon |
| 11:15 AM | 12:30 PM | **T06** Windows 10—The Universal Application: One App To Rule Them All? - *Laurent Bugnion* | **T07** TypeScript for C# Developers - *Chris Klug* | **T08** Using Visual Studio Tools for Apache Cordova to Create Multi-Platform Applications - *Kevin Ford* | **T09** No Schema, No Problem! – Introduction to Azure DocumentDB - *Leonard Lobel* | **T10** This Microsoft session is sequestered, details will be released soon |
| 12:30 PM | 2:00 PM | Lunch • Visit Exhibitors | | | | |
| 2:00 PM | 3:15 PM | **T11** Strike Up a Conversation with Cortana on Windows 10 - *Walt Ritscher* | **T12** Angular 2 Forms and Validation - *Deborah Kurata* | **T13** Create Great Looking Android Applications using Material Design - *Kevin Ford* | **T14** Dependencies Demystified - *Jason Bock* | **T15** This Microsoft session is sequestered, details will be released soon |
| 3:15 PM | 3:45 PM | Sponsored Break • Visit Exhibitors | | | | |
| 3:45 PM | 5:00 PM | **T16** Building Truly Universal Applications - *Laurent Bugnion* | **T17** Debugging the Web with Fiddler - *Ido Flatow* | **T18** Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - *Nick Landry* | **T19** Improving Performance in .NET Applications - *Jason Bock* | **T20** This Microsoft session is sequestered, details will be released soon |
| 5:00 PM | 6:30 PM | Microsoft Ask the Experts & Exhibitor Reception • Attend Exhibitor Demos | | | | |
| **START TIME** | **END TIME** | **Visual Studio Live! Day 2: Wednesday, August 10, 2016** | | | | |
| 8:00 AM | 9:15 AM | **W01** Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - *Benjamin Day* | **W02** Busy JavaScript Developer's Guide to ECMAScript 6 - *Ted Neward* | **W03** Windows for Makers: Raspberry Pi, Arduino & IoT - *Nick Landry* | **W04** Applying S.O.L.I.D. Principles in .NET/C# - *Chris Klug* | **W05** ASP.NET Core: Reimagining Web Application Development in .NET - *Ido Flatow* |
| 9:30 AM | 10:45 AM | **W06** Team Foundation Server 2015: Must-Have Tools and Widgets - *Richard Hundhausen* | **W07** Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - *Benjamin Day* | **W08** Build Cross-Platform Mobile Apps with Ionic, Angular, and Cordova - *Brian Noyes* | **W09** Open Source Software for Microsoft Developers - *Rockford Lhotka* | **W10** This Microsoft session is sequestered, details will be released soon |
| 11:00 AM | 12:00 PM | **GENERAL SESSION**: To Be Announced | | | | |
| 12:00 PM | 1:30 PM | Birds-of-a-Feather Lunch • Visit Exhibitors | | | | |
| 1:30 PM | 2:45 PM | **W11** Stop the Waste and Get Out of (Technical) Debt - *Richard Hundhausen* | **W12** Getting Started with Aurelia - *Brian Noyes* | **W13** Top 10 Entity Framework Features Every Developer Should Know - *Philip Japikse* | **W14** Creating Dynamic Pages Using MVVM and Knockout.JS - *Christopher Harrison* | **W15** This Microsoft session is sequestered, details will be released soon |
| 2:45 PM | 3:15 PM | Sponsored Break • Exhibitor Raffle @ 2:55 pm (Must be present to win) | | | | |
| 3:15 PM | 4:30 PM | **W16** Real World VSTS Usage for the Enterprise - *Jim Szubryt* | **W17** What You Need To Know About ASP.NET Core and MVC 6 - *Rachel Appel* | **W18** Predicting the Future Using Azure Machine Learning - *Eric D. Boyd* | **W19** This session is sequestered, details will be released soon | **W20** This Microsoft session is sequestered, details will be released soon |
| 6:45 PM | 9:00 PM | Set Sail! VSLive's Seattle Sunset Cruise on Lake Washington | | | | |
| **START TIME** | **END TIME** | **Visual Studio Live! Day 3: Thursday, August 11, 2016** | | | | |
| 8:00 AM | 9:15 AM | **TH01** Enterprise Reporting from VSTS - *Jim Szubryt* | **TH02** Build Real-Time Websites and Apps with SignalR - *Rachel Appel* | **TH03** Write Better C# - *James Montemagno* | **TH04** Pretty, Yet Powerful. How Data Visualization Transforms the Way we Comprehend Information - *Walt Ritscher* | **TH05** Docker and Azure - *Steve Lasker* |
| 9:30 AM | 10:45 AM | **TH06** App Services Overview —Web, Mobile, API and Logic Oh My... - *Mike Benkovich* | **TH07** Breaking Down Walls with Modern Identity - *Eric D. Boyd* | **TH08** Debugging Your Way Through .NET with Visual Studio 2015 - *Ido Flatow* | **TH09** Windows 10 Design Guideline Essentials - *Billy Hollis* | **TH10** This Microsoft session is sequestered, details will be released soon |
| 11:00 AM | 12:15 PM | **TH11** Real-World Azure DevOps - *Brian Randell* | **TH12** AngularJS & ASP.NET MVC Playing Nice - *Miguel Castro* | **TH13** Build Distributed Business Apps Using CSLA .NET - *Rockford Lhotka* | **TH14** Learning to Live Without Data Grids in Windows 10 - *Billy Hollis* | **TH15** This Microsoft session is sequestered, details will be released soon |
| 12:15 PM | 2:15 PM | Lunch @ The Mixer • Visit the Microsoft Company Store & Visitor Center | | | | |
| 2:15 PM | 3:30 PM | **TH16** Cloud Oriented Programming - *Vishwas Lele* | **TH17** Richer MVC Sites with Knockout JS - *Miguel Castro* | **TH18** PowerShell for Developers - *Brian Randell* | **TH19** Take Your Site From Ugh to OOH with Bootstrap - *Philip Japikse* | **TH20** This Microsoft session is sequestered, details will be released soon |
| 3:45 PM | 5:00 PM | **TH21** Migrating Cloud Service Applications to Service Fabric - *Vishwas Lele* | **TH22** Test Drive Automated GUI Testing with WebDriver - *Rachel Appel* | **TH23** Tour d'Azure 2016 Edition...New Features and Capabilities that Dev's Need to Know - *Mike Benkovich* | **TH24** Busy Developers Guide to Node.js - *Ted Neward* | **TH25** This Microsoft session is sequestered, details will be released soon |
| **START TIME** | **END TIME** | **Visual Studio Live! Post-Conference Workshops: Friday, August 12, 2016** *(Separate entry fee required)* | | | | |
| 8:00 AM | 5:00 PM | **F01** Workshop: Data-Centric Single Page Apps with Angular 2, Breeze, and Web API - *Brian Noyes* | | **F02** Workshop: Building Business Apps on the Universal Windows Platform - *Billy Hollis* | | |

***Speakers and sessions subject to change***

**CONNECT WITH VISUAL STUDIO LIVE!**

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

**VSLIVE.COM/REDMOND**

# Enterprise Application Integration Using Azure Logic Apps

Srikantan Sankaran

Business process management is a key focus area for many enterprises and integration with heterogeneous systems forms the heart of process automation. Under this context, a critical use case that emerges is related to ingestion, transformation and routing of data across the enterprise and the Microsoft Azure Cloud. Using the traditional approach to implement such enterprise application integration (EAI) scenarios entails a significant investment of time and cost—to identify and invest in a tool, to build technical expertise and to provision compute infrastructure to support the scenarios end-to-end. A cloud-based managed service like Azure Logic Apps addresses these problem areas.

The Logic Apps feature of the Azure Apps Service provides a visual editing experience where developers can compose complex data exchange between multiple systems or data sources using a suite of standard and enterprise integration connectors. Scenarios like handling of transient failure, which is unique to cloud-based architectures, and others like condition-based routing of process flows and long-running transactions, are easily implemented through this service.

The capabilities of Logic Apps are demonstrated in this article through an enterprise integration scenario where a seller organization has deployed an order capturing system on Azure that leverages the capabilities provided by Logic Apps. This Logic App process picks up orders from a secure public endpoint exposed by a buyer organization, which is then passed through numerous steps comprised of data capture, data transformation and routing prior to getting registered in the back-end system. An acknowledgement ID is generated and sent back to the buyer organization for each of the orders registered.

This article discusses:

- Creating an Azure Logic App flow using standard and enterprise connectors
- Using Visual Studio 2012 with BizTalk Services to author Schemas and Transformation Maps, and consume them in the Logic App Flow
- Implementing condition-based routing of workflow process, retrying logic to handle transient failures and working with the Code Editor in a Logic App
- Extending Logic Apps capabilities to address Hybrid scenarios in integration

Technologies discussed:

Azure Logic Apps, BizTalk Services, Visual Studio 2012, Hybrid Connections, Logic App Connectors

Code download available at:

bit.ly/1poJPgs

## EAI Process Flow

**Figure 1** depicts a simple EAI scenario that's implemented using Logic Apps.

Order data that triggers the integration is uploaded from the buyer organization in the form of flat files that are exposed securely

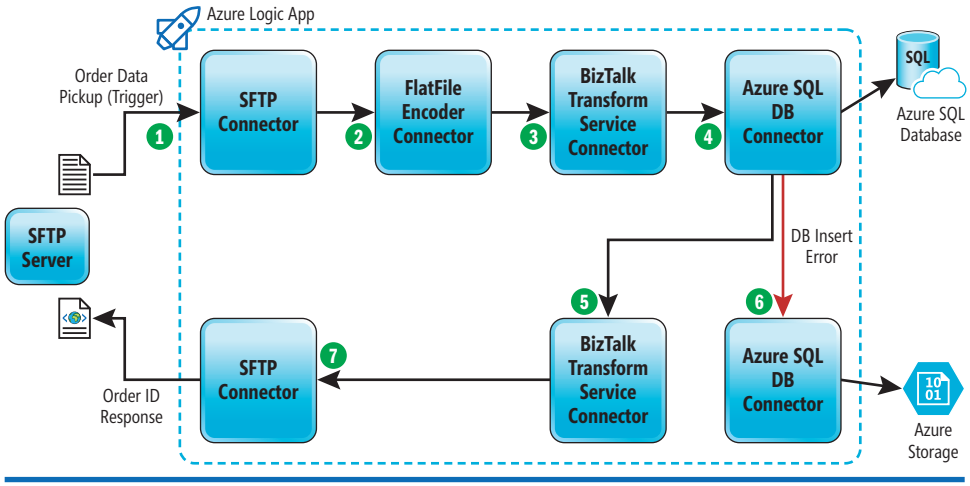



Figure 1 **Enterprise Application Integration Process Flow**

using an SSH File Transfer Protocol (SFTP) endpoint over the Internet. Access to the endpoint is secured with authentication credentials that are shared with the seller organization. Separate folders are created to drop order data into and to receive acknowledgement messages. An Azure virtual machine running CentOS Linux is used as the SFTP server that hosts the endpoint for the file pickup and drop.

An Azure SFTP connector is used to connect to the SFTP endpoint from where the order data is picked up based on a time-triggered frequency. An instance of this connector is created in the Azure portal, from the Azure Marketplace and deployed as an API App. The same API App instance would also be used when uploading the order acknowledgement data back to the SFTP endpoint.

Here are key parameters to set in the SFTP connector instance configuration, when added to the Logic App flow:

- Trigger frequency: 2 minutes used in the scenario implementation
- Pickup location: Add a path relative to the *root path (*for example, */home/<sftpusername>)* configured in the SFTP connector instance earlier; in this case: b2b/orders
- Select the Delete File after Read option

The BizTalk FlatFile encoder is used to read the flat file from the SFTP connector API App and convert that into a schema-bound XML document. The FlatFile Encoder is available as part of the Enterprise Integration Connectors for Logic Apps. An instance of this connector is created from the Azure Marketplace and deployed as an API App before consumption in the Logic App flow. While the FlatFile Encoder supports basic XSD schema generation from Flat files and JSON Documents, for more complex requirements involving multiple child records, such as those used in this scenario, the schema are generated using Visual Studio 2012 and BizTalk Services SDK. The XSD schema documents are then uploaded to the BizTalk FlatFile Encoder Connector API App.

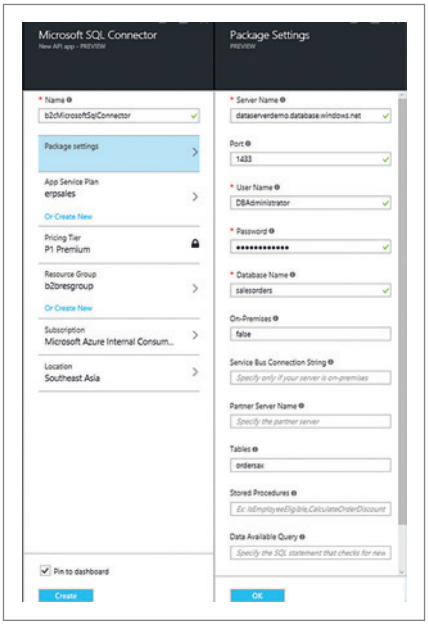



Figure 2 **Azure SQL Database Connector Configuration**

The BizTalk Transform Services connector converts the XML document containing the order data into an XML payload for insertion into the Order Capture Database. It uses a Transformation Map that's authored using Visual Studio 2012 and the BizTalk Services SDK. To the instance of this connector that's deployed as an API App, the Map files, both for the order registration and for the acknowledgement flow, are uploaded. The schema files used in the Map do not have to be uploaded to this API App.

The SQL Database Connector API App instance consumes the XML document payload from the previous step and inserts the order data into the Azure SQL Database. The schema used to generate the XML payload for the insert operation can be generated in Visual Studio 2012 with BizTalk Services, based on an XML document instance, or using the BizTalk WCF LOB Adapters for SQL Server. However, the Azure SQL Database Connector API App itself now supports the generation of the schema for all the operations on a database, directly from the API App configuration in the Azure Portal. This obviates the need for the additional tools like the BizTalk WCF LOB Adapters or other third-party solutions, which otherwise would have been required for on-premises deployments.

**Figure 2** shows some of the key parameters to set in the Azure SQL Database Connector.

On successful insertion of order data, the response from the Database Connector is transformed using the BizTalk Transform Service Connector API App configured in the earlier steps, from an Array of Order ID numbers to a comma-separated list.

The Azure Storage Blob Connector API App Instance is used in the Logic App flow to store the error messages when the database insertion in the previous step fails.

The order IDs generated in the response are uploaded using the SFTP Connector configured earlier to the buyer organization's SFTP endpoint.

The Logic App process flow, when complete, would appear as depicted in **Figure 3**.

At this time, the default schema version that's used when creating a new Logic App is 2015-08-01-preview, which does not yet provide the ability to add enterprise connectors. Hence, to implement the current EAI scenario, this schema version should be replaced with 2014-12-01-preview. This can be done in the Code View of the Logic App by editing the JSON code directly.

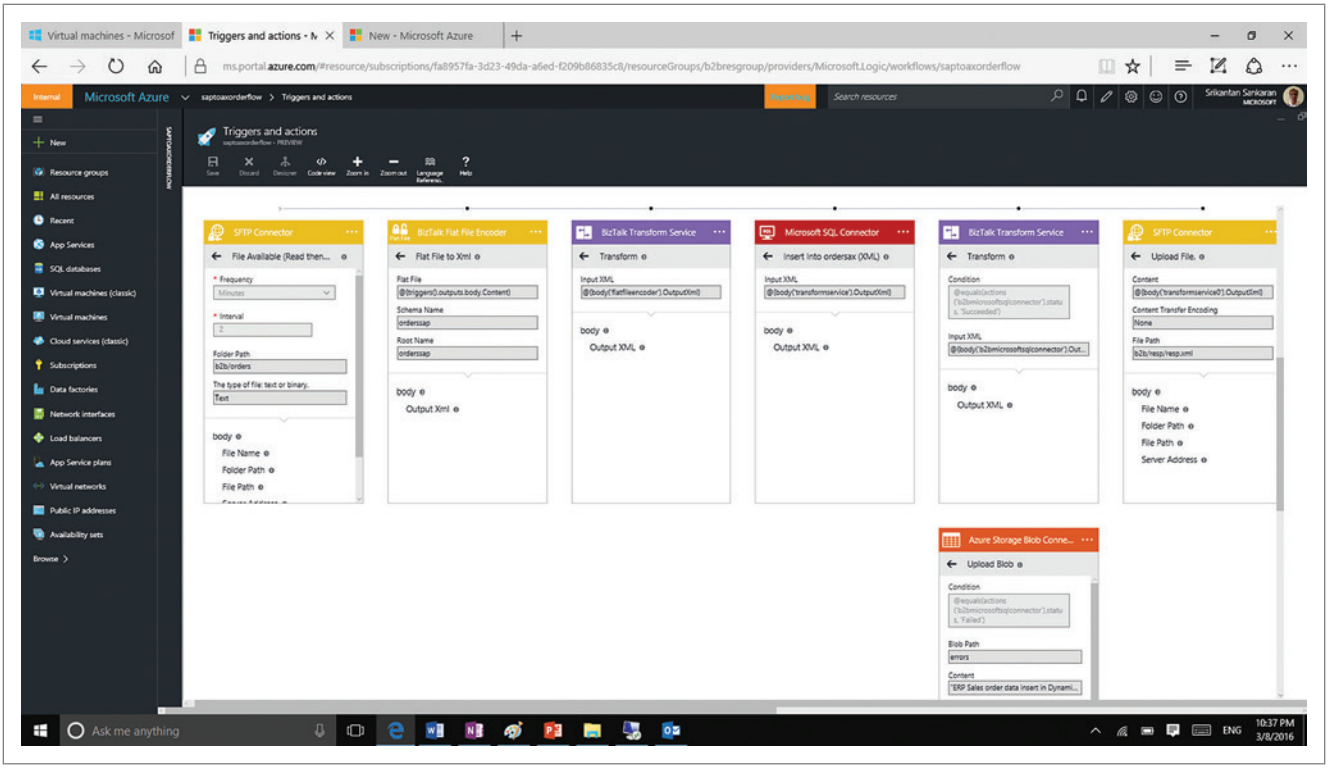



Figure 3 **Complete Logic App Flow**

Remember to save changes, close the Logic App editor and reload it to let the changes take effect.

When adding resources into the Logic App flow like all the connectors, Azure SQL Database, Storage Connections and so on, it's good practice to group all of them under a single resource group for ease of handling and ease of deployment. It's also preferable that the resources are deployed in the same region (Azure Datacenter) in order to minimize latency during the flow execution.

## Transient Error Handling in the Flow

Azure Logic Apps provide the ability to incorporate a retry logic to take care of transient errors that are unique to cloud-based architectures. This involves adding a JSON code fragment—as you can see by the highlighted code in **Figure 4**.

Figure 4 **Handling Transient Failures**

```
"b2bmicrosoftsqlconnector": {
  "type": "ApiApp",
  "inputs": {
    "apiVersion": "2015-01-14",
    "host": {

      },
    "operation": "XMLInsertOnordersax",
    "parameters": {
      "requestMessage": {
        "InputXml": "@{body('transformservice').OutputXml}"
      }
    },
    "retryPolicy" : {
      "type": "fixed",
      "interval": "PT30S",
      "count": 2
    },
  },
```

When there are intermittent failures or connectivity exceptions in executing the order insert operation in the Azure SQL Database, the Connector API App will retry the request two times until it returns an error.

> Azure Logic Apps provide the ability to incorporate retry logic to take care of transient errors that are unique to the cloud-based architectures.

Logic Apps also support the ability to add features such as Repeating Call execution until a predetermined condition is met, implement Wait Actions, invoke other workflows and so on.

## Conditional Logic App Flow Execution

Within a Logic App flow, it's possible to define actions based on the outcome of the previous operation. The option to define a condition can be set either in the Configuration pane in the Designer or in the Code View of the Logic App.

For example, in the current flow, if the outcome of the order insertion into the Azure SQL Database is successful, then the response Message is transformed before upload to the SFTP endpoint at the buyer organization.

# Spreadsheets Made Easy.

## Fastest Calculations

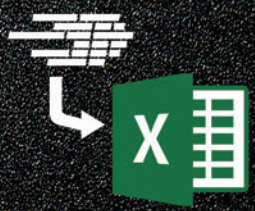
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.


## Powerful Controls

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

## Comprehensive Charting

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




## Scalable Reporting

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

Figure 5 **XML File Used to Generate XSD Schema for Azure SQL Database Order Data Insert**

```
<Insert xmlns="http://schemas.microsoft.com/Sql/2008/05/TableOp/dbo/ordersax ">
  <Rows>
  <ordersax XMLns="http://schemas.microsoft.com/Sql/2008/05/Types/Tables/dbo">
    <orderidsap>sap0101</orderidsap>
    <orderdate>2016-02-03</orderdate>
    <supplierid>sail001</supplierid>
    <orderamount>25000</orderamount>
    <currency>INR</currency>
    <discount>12</discount>
    <delvydate>2016-06-03</delvydate>
    <contract>C001</contract>
    <contact>Mr</contact>
  </ordersax>
  <ordersax xmlns="http://schemas.microsoft.com/Sql/2008/05/Types/Tables/dbo">
    <...>
  </ordersax>
... and more records ...
  </Rows>
</Insert>
```

In case of a failure, an error message is uploaded to the Azure Storage using the Azure Blob Storage Connector. The Logic App UI visually depicts the alternate flows thus configured. Refer to **Figure 6** that depicts how Conditional Actions can be defined and how the Logic App visually depicts the flow that involves them.

## Working with XSD Schema Documents

Development tools that developers have been using to build their artifacts for BizTalk Server-based Enterprise Integration Solutions can be extended for use to Logic Apps, as well. The BizTalk Services SDK, in conjunction with Visual Studio 2012, provides additional project templates that enable the developer to generate or author XSD schemas and transformation maps. There are built-in wizards that could be used to generate these both from flat files and from XML document instances. The output XSD schema documents generated in Visual Studio 2012 could be directly uploaded to the FlatFile Encoder Instance used in the Logic App.

A sample XML file that was used to generate the XSD schema for Azure SQL Database order data insert, using the Schema Generation Wizard in Visual Studio 2012, is shown in **Figure 5**.

The reusability of such artifacts that are used for on-premises deployment of EAI Solutions using BizTalk Server provides enterprises extended value on their investments to explore Azure as an option on which to deploy their solutions. Along with that, there are additional benefits like quick time to deployment, ease of provisioning of infrastructure, ease of manageability and monitoring of the solution.

## Working with Maps

The Azure BizTalk transform service connector uses BizTalk transform maps to transform XML documents. However, to create these maps, Visual Studio 2012 with BizTalk Services SDK is required. As with the XSD schemas, the map transform files once generated here can be imported into Azure Logic Apps directly.

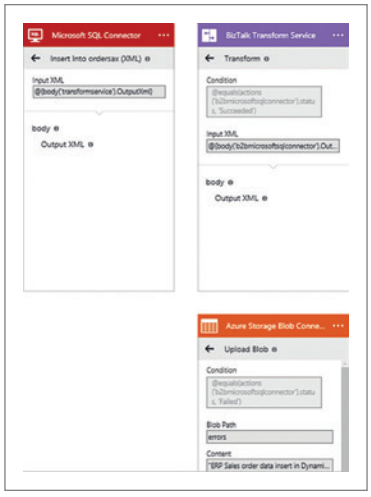



Figure 6 **Visual Representation of Conditional Actions**

In the Process Flow considered here, to map the incoming order XML to the database Insert XML schema and to iterate through every record in the incoming document, the MapEach Loop functoid is used, as depicted in **Figure 7**.

To map the order creation response message to the outgoing message, the array of Order IDs generated from the SQL Server database needs to be converted into comma-separated values inside a single Element. To achieve this, the Cumulative Concatenate functoid is used.

## Executing the Scenario

In order to try out the scenario described here, drop the file orderssap.csv into the /home/<sftpusername>/b2b/orders folder using the WinSCP/equivalent Tool.

Monitor the status of the Logic App Connector trigger logs to ensure that the trigger has indeed fired.

If there are no errors, a file, resp.XML, would be deposited in the /home/<sftpusername>/b2b/resp folder, which would contain the comma-separated order ID values generated from the Order Processing System in Azure.

> Using Hybrid Connections, the use of the BizTalk Enterprise Connectors can be extended to line-of-business systems like SAP and SharePoint deployed inside corporate networks.

Now, drop the orderssap.csv file again into the same folder, without any changes. The Database Connector will throw an exception due to a Unique Constraint Violation error from the database. An error message will be uploaded into Azure Blob Storage, into the container "orders."

## Logic App Execution—Tracking and Monitoring

Use the Operations tile to monitor the execution of the Logic App Flow. There's also a separate log for the trigger events on the Logic App. Drill down into each of them and view a breakdown of the execution of each connector in the flow, the outcome of the execution and duration.

## Extending the Scenario for Cross-Premises Integration

While the SQL database consumed in this scenario is deployed in Azure, it would be just as easy to integrate with a SQL server database

running inside a corporate network. The Azure App Services infrastructure, of which Logic Apps is a part, provides a feature called hybrid connections. Implementation of this feature involves installation of a Hybrid Connection Manager on a server behind the firewall in the corporate network, through which the Logic App integrates with this database. Apart from SQL Server, other databases that can be accessed using hybrid connections are Oracle, DB2 and Informix.

Using hybrid connections, the use of the BizTalk enterprise connectors can be extended to line-of-business (LOB) systems like SAP and SharePoint deployed inside corporate networks. They can also be extended to REST Services and Web services deployed on premises.

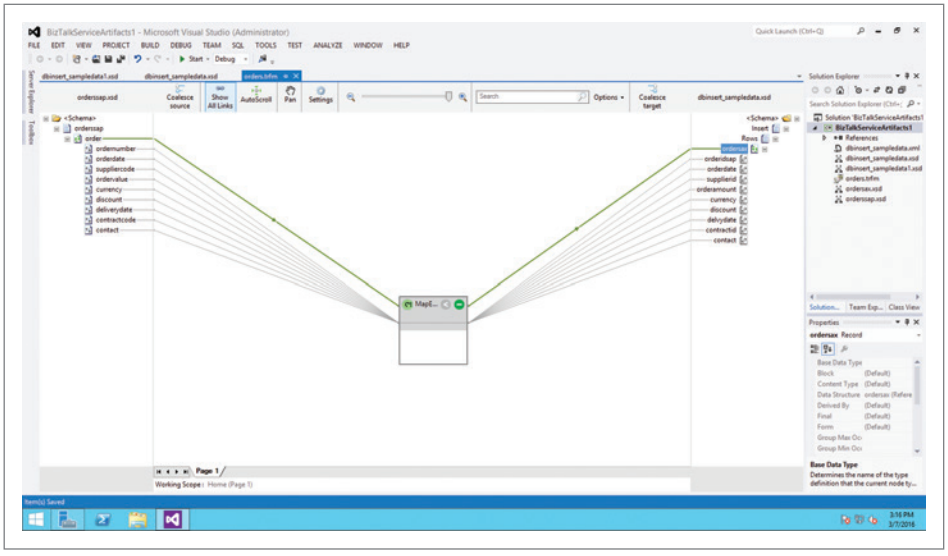


Figure 7 **Mapping of Order Insert Acknowledgement**

## Software Prerequisites to Implement This Scenario

Apart from an Azure subscription that's required to implement the scenario described here, the following development tools and additional software is also required:

- Visual Studio 2012 along with the BizTalk Services SDK running on Windows 8/8.1/10 or Windows Server 2012 R2 is required.
- SQL Management Studio for SQL Server 2014 or later to create and connect to the Azure SQL Database. Alternatively, use Visual Studio 2012/or higher Tools for SQL Server.
- WinSCP or an equivalent tool to connect to the SFTP Server, create the "send" and "receive" folders and to drop and receive messages.

## Artifacts Available for Download

The following artifacts used to implement this scenario are available for download from the GitHub repository at bit.ly/1poJPgs:

- Azure SQL Database creation script–ordersax.sql
- Sample order data file–orderssap.csv
- XSD schemas:
  - dbinsert_sampledata.xsd: SQL Insert schema
  - dbinsert_sampledata1.xsd: part of, and referenced in the schema above
  - orderssap.xsd: schema for the incoming Order Document
  - orderresponse.xsd: SQL Insert response
  - orderresponse1.xsd: part of, and referenced in the SQL Insert response schema
  - ordersaxresp.xsd: schema of the outgoing XML response message
- Maps:
  - orders.trfm: Mapping incoming message to SQL Insert schema
  - ordersaxtosap.trfm: Mapping SQL Insert response schema to outgoing Message schema

The following references would be useful when implementing the solution covered in this scenario:

- Software prerequisites to develop schemas and maps used in the solution: bit.ly/1R3RzvH
- Using Visual Studio 2012 to generate schemas from flat files: bit.ly/1nP6qBy
- Configuring an SFTP Connector in Azure: bit.ly/254QfC2

## Wrapping Up

Azure Logic Apps provides a platform to implement end-to-end EAI scenarios. The comprehensive set of standard connectors and enterprise connectors makes it possible to integrate with numerous enterprise LOB systems and Software-as-a-Service applications in the cloud. It can seamlessly consume artifacts generated for on-premises EAI solution deployments using BizTalk Server 2013, like schemas and maps. This provides enterprises with an option to repurpose their current investments and unlock their potential to implement scenarios in the cloud, leveraging the benefits of high availability, reliability and ease of set up of the infrastructure that Azure provides. With hybrid connections support in Logic Apps, additional value can be derived from current investments made in LoB systems deployed on-premises, by opening up opportunities to have them participate in integration scenarios running in the cloud. ■

## Tip!

**A quick way** to get started with developing the artifacts in the solution is to use the Gallery VM image in the Azure Management Portal that runs BizTalk Server 2013 Standard Edition. Installing Visual Studio 2012 and the BizTalk Services SDK in the VM took me no more than 30 minutes.

**Srikantan Sankaran** *is a technical evangelist from the DX team in India, based out of Bangalore. He works with numerous ISVs in India and helps them architect and deploy their solutions on Microsoft Azure. Reach him at sansri@microsoft.com.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Sandeep Alur*

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

# Orlando 2016

ROYAL PACIFIC RESORT AT UNIVERSAL
DECEMBER 5-9


## The Ultimate Education Destination

**Live! 360℠ is a unique conference** where the IT and Developer community converge to debate leading edge technologies and educate themselves on current ones. These six 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.
**Choose the ultimate education destination: Live! 360.**

EVENT PARTNERS
Microsoft Magenic

PLATINUM SPONSOR
GRIDSTORE

GOLD SPONSOR
GOVERLAN COMPLEX IT SYSTEMS MANAGEMENT MADE EASY

SUPPORTED BY
Visual Studio msdn magazine Redmond Channel Partner

# 6 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 TODAY AND **SAVE $500!**

Use promo code L360MAY2

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

# Take the Tour

## Visual Studio Live!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

**Visual Studio Live!:** VSLive!™ is a victory for code, featuring unbiased and practical development training on the Microsoft Platform.

## SQL Server Live!

TRAINING FOR DBAs AND IT PROS

**SQL Server Live!:** This conference will leave you with the skills needed to Lead the Data Race, whether you are a DBA, developer, IT Pro, or Analyst.

## TechMentor

IN-DEPTH TRAINING FOR IT PROS

**TechMentor℠:** This is IT training that finishes first, with zero marketing speak on topics you need training on now, and solid coverage on what's around the corner.

## Office & SharePoint Live!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

**Office & SharePoint Live!:** Provides leading-edge knowledge and training for SharePoint both on-premises and in Office 365 to maximize the business value.

## Modern Apps Live!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

**Modern Apps Live!:** Presented in partnership with Magenic, this unique conference leads the way to learning how to architect, design and build a complete Modern App.

## APPDEV TRENDS

ENTERPRISE FOCUSED. CODE DRIVEN.

NEW!

**App Dev Trends:** This new technology conference focuses on the makers & maintainers of the software that Power organizations in nearly every industry in the world – in other words, enterprise software professionals!

Redmond Magazine · Virtualization Review · Visual Studio Magazine

PRODUCED BY 1105 MEDIA

LIVE360EVENTS.COM



JAMES MCCAFFREY

# The Multi-Armed Bandit Problem

Imagine you're in Las Vegas, standing in front of three slot machines. You have 20 tokens to use, where you drop a token into any of the three machines, pull the handle and are paid a random amount. The machines pay out differently, but you initially have no knowledge of what kind of payout schedules the machines follow. What strategies can you use to try and maximize your gain?

This is an example of what's called the multi-armed bandit problem, so named because a slot machine is informally called a one-armed bandit. The problem is not as whimsical as it might first seem. There are many important real-life problems, such as drug clinical trials, that are similar to the slot machine example.

It's unlikely you'll ever need to code an implementation of the multi-armed bandit problem in most enterprise development scenarios. But you might want to read this article for three reasons. First, several of the programming techniques used in this article can be used in other, more common programming scenarios. Second, a concrete code implementation of the multi-armed bandit problem can serve as a good introduction to an active area of economics and machine learning research. And third, you just might find the topic interesting for its own sake.

The best way to get a feel for where this article is headed is to take a look at the demo program shown in **Figure 1**. There are many different algorithms that can be used on multi-armed bandit problems. For example, a completely random approach would be to just select a machine at random for each pull, then hope for the best. The demo presented here uses a basic technique called the explore-exploit algorithm.

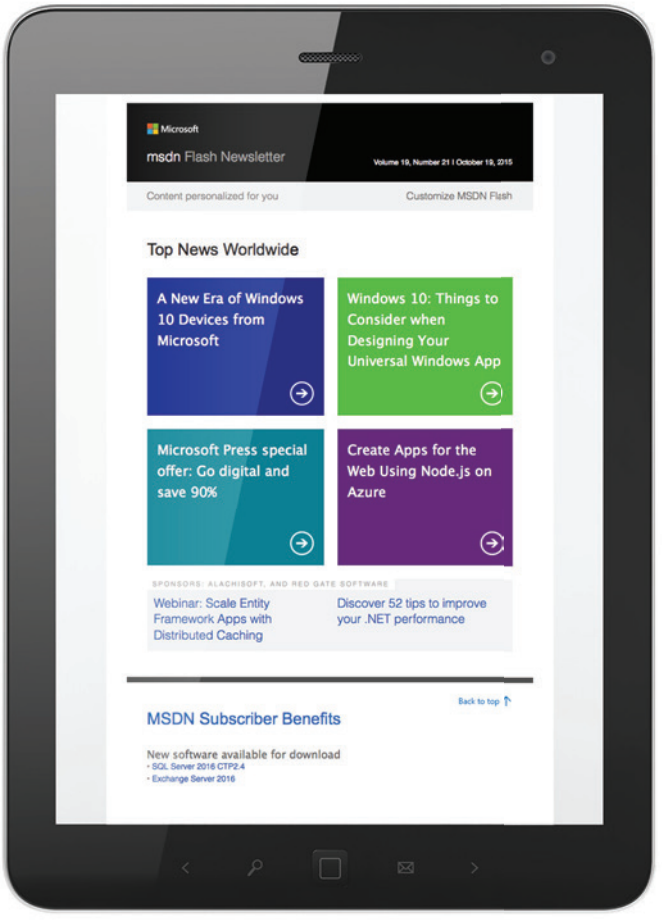
The demo begins by creating three machines. Each machine pays a random amount on each pull, where the payout follows a Gaussian (bell-shaped) distribution with a specified mean (average) and standard deviation. The third machine is the best in one sense because it has the highest mean payout per pull of 0.1 arbitrary units. In a non-demo scenario, you wouldn't know the characteristics of the machines.

The total number of pulls available is set to 20. In explore-exploit, you set aside a certain proportion of your allotted pulls and use them to try and find the best machine. Then you use your remaining pulls only on the best machine found during the preliminary explore phase. The key variable for the explore-exploit algorithm is the percentage of pulls you designate for the explore phase. The demo sets the explore-percentage to 0.40, therefore, there are 20 * 0.40 = 8 explore pulls followed by 20 - 8 = 12 exploit pulls. Increasing the percentage of explore pulls increases the probability of finding the best machine, at the expense of having fewer pulls left to take advantage of the best machine in the exploit phase.

```
file:///C:/MultiBandit/bin/Debug/MultiBandit....

Begin multi-armed bandit problem demo

Creating 3 Gaussian machines
Machine 0 mean =  0.0, sd = 1.0
Machine 1 mean = -0.5, sd = 2.0
Machine 2 mean =  0.1, sd = 0.5

Best machine is [2] with mean pay = 0.1
Setting nPulls = 20

Using explore-exploit with pctExplore = 0.40

Start explore phase
[  0]  selected machine 2. pay =   0.19
[  1]  selected machine 1. pay =  -0.46
[  2]  selected machine 0. pay =  -0.09
[  3]  selected machine 2. pay =   0.14
[  4]  selected machine 0. pay =   0.12
[  5]  selected machine 0. pay =   0.29
[  6]  selected machine 2. pay =   0.70
[  7]  selected machine 1. pay =  -1.91

Best machine found during explore = 2

Start exploit phase
[  0]  pay =   0.30
[  1]  pay =   0.33
[  2]  pay =   0.23
[  3]  pay =   0.68
[  4]  pay =  -0.49
[  5]  pay =  -0.15
[  6]  pay =  -0.03
[  7]  pay =   0.74
[  8]  pay =  -0.09
[  9]  pay =   0.36
[ 10]  pay =  -0.01
[ 11]  pay =   0.45

Average pay per pull = 0.06
Total payout         = 1.30

Baseline average pay = 0.10
Total baseline pay   = 2.00

Total regret = 0.70

End bandit demo
```

Figure 1 **Using Explore-Exploit on a Multi-Armed Bandit Problem**

Code download available at msdn.com/magazine/0516magcode.




# Get news from MSDN in your inbox!

Sign up to receive **MSDN FLASH**, which delivers the latest resources, SDKs, downloads, partner offers, security news, and updates on national and local developer events.

msdn magazine

**msdn.microsoft.com/flashnewsletter**

During the eight-pull explore phase, the demo displays which machine was randomly selected and the associated payout. Behind the scenes, the demo saves the accumulated payouts of each machine. Machine 0 was selected three times and paid -0.09 + 0.12 + 0.29 = +0.32 units. Machine 1 was selected two times and paid -0.46 + -1.91 = -2.37 units. Machine 2 was selected three times and paid 0.19 + 0.14 + 0.70 = +1.03 units. In the case of the demo, the explore-exploit algorithm correctly identifies machine 2 as the best machine because it has the largest total payout. At this point the algorithm has a net gain (loss) of 0.32 + -2.37 + 1.03 = -1.02 units.

During the 12-pull exploit phase, the demo repeatedly plays only machine 2. The exploit phase payouts are 0.03 + 0.33 + . . + 0.45 = +2.32 units. Therefore, the total payout over all 20 pulls is -1.02 + 2.32 = +1.30 units and the average payout per pull is 1.30 / 20 = 0.065.

There are several different metrics that can be used to evaluate the effectiveness of a multi-armed bandit algorithm. One common measure is called regret. Regret is the difference between a theoretical baseline total payout and the actual total payout for an algorithm. The baseline theoretical payout is the expected payout if all allotted pulls were used on the best machine. In the case of the three machines in the demo, the best machine has an average payout of 0.10 units, so the expected payout if all 20 pulls were used on that machine is 20 * 0.10 = 2.00 units. Because the explore-exploit algorithm yielded a total payout of only 1.30 units the regret metric is 2.00 - 1.30 = 0.70 units. Algorithms with lower regret values are better than those with higher regret values.

This article assumes you have at least intermediate programming skills, but doesn't assume you know anything about the multi-armed bandit problem. The complete demo program, with a few minor edits to save space, is presented in **Figure 2**, and it's also available in the associated code download. The demo is coded using C#, but you shouldn't have too much trouble refactoring the demo to another language, such as Python or Java. All normal error checking was removed from the demo in order to keep the main ideas of the multi-armed bandit problem as clear as possible.

## The Demo Program

To create the demo program, I launched Visual Studio and selected the C# console application template. I named the project MultiBandit. I used Visual Studio 2015, but the demo has no significant .NET version dependencies so any version of Visual Studio will work.

After the template code loaded, in the Solution Explorer window I right-clicked on file Program.cs and renamed it to the more descriptive MultiBanditProgram.cs, then allowed Visual Studio to automatically rename class MultiBandit. At the top of the code in the editor window, I deleted all unnecessary using statements, leaving just the one reference to the top-level System namespace.

All the control logic is in the Main method, which calls method ExploreExploit. The demo has a program-defined Machine class, which in turn has a program-defined nested class named Gaussian.

After some introductory WriteLine statements, the demo creates three machines:

```
int nMachines = 3;
Machine[] machines = new Machine[nMachines];
machines[0] = new Machine(0.0, 1.0, 0);
machines[1] = new Machine(-0.5, 2.0, 1);
machines[2] = new Machine(0.1, 0.5, 2);
```

The Machine class constructor accepts three arguments: the mean payout, the standard deviation of the payouts and a seed for random number generation. So, machine [1] will pay out -0.5 units per pull on average, where most of the payouts (roughly 68 percent) will be between -0.5 - 2.0 = -1.5 units and -0.5 + 2.0 = +1.5 units. Notice that unlike real slot machines, which pay out either zero or a positive amount, the demo machines can pay out a negative amount.

The statements that perform the explore-exploit algorithm on the three machines are:

```
int nPulls = 20;
double pctExplore = 0.40;
double avgPay = ExploreExploit(machines, pctExplore, nPulls);
double totPay = avgPay * nPulls;
```

Method ExploreExploit returns the average gain (or loss if negative) per pull after nPulls random events. Therefore, the total pay from the session is the number of pulls times the average pay per pull. An alternative design is for ExploreExploit to return the total pay instead of the average pay.

The regret is calculated like so:

```
double avgBase = machines[2].mean;
double totBase = avgBase * nPulls;
double regret = totBase - totPay;
```

Variable avgBase is the average payout per pull of the best machine, machine [2] = 0.1 units. So the total average expected payout over two pulls is 20 * 0.10 = 2.0 units.

## Generating Gaussian Random Values

As I mentioned, each machine in the demo program pays out a value that follows a Gaussian (also called normal, or bell-shaped) distribution. For example, machine [0] has a mean payout of 0.0 with a standard deviation of 1.0 units. Using the code from the demo that generates Gaussian values, I wrote a short program to produce 100 random payouts from machine [0]. The results are shown in the graph in **Figure 3**.

Notice that the majority of generated values are close to the mean. The variability of the generated values is controlled by the value of the standard deviation. A larger standard deviation produces a larger spread of values. In a multi-armed bandit problem, one of the most important factors for all algorithms is the variability of machine payouts. If a machine has highly variable payouts, it becomes very difficult to evaluate the machine's true average payout.

There are several algorithms that can be used to generate Gaussian distributed random values with a specified mean and standard deviation. My preferred method is called the Box-Muller algorithm. The Box-Muller algorithm first generates a uniformly distributed value (the kind produced by the .NET Math.Random class) and then uses some very clever mathematics to transform the uniform value into one that's Gaussian distributed. There are several variations of Box-Muller. The demo program uses a variation that's somewhat inefficient compared to other variations, but very simple.

In the demo program, class Gaussian is defined inside class Machine. In the Microsoft .NET Framework, nested class definitions are mainly a convenience for situations where the nested class is a utility class used by the outer containing class. If you're porting this demo code to a non-.NET language, I recommend refactoring class Gaussian to a standalone class. The Gaussian class has a single constructor that accepts a mean payout, a standard deviation for the payout and a seed value for the underlying uniform random number generator.

Figure 2 **Complete Multi-Armed Bandit Demo Code**

```
using System;
namespace MultiBandit
{
  class MultiBanditProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("\nBegin multi-armed bandit demo \n");

      Console.WriteLine("Creating 3 Gaussian machines");
      Console.WriteLine("Machine 0 mean =  0.0, sd = 1.0");
      Console.WriteLine("Machine 1 mean = -0.5, sd = 2.0");
      Console.WriteLine("Machine 2 mean =  0.1, sd = 0.5");
      Console.WriteLine("Best machine is [2] mean pay = 0.1");

      int nMachines = 3;
      Machine[] machines = new Machine[nMachines];
      machines[0] = new Machine(0.0, 1.0, 0);
      machines[1] = new Machine(-0.5, 2.0, 1);
      machines[2] = new Machine(0.1, 0.5, 2);

      int nPulls = 20;
      double pctExplore = 0.40;
      Console.WriteLine("Setting nPulls = " + nPulls);

      Console.WriteLine("\nUsing pctExplore = " +
        pctExplore.ToString("F2"));
      double avgPay = ExploreExploit(machines, pctExplore,
        nPulls);
      double totPay = avgPay * nPulls;
      Console.WriteLine("\nAverage pay per pull = " +
        avgPay.ToString("F2"));
      Console.WriteLine("Total payout         = " +
        totPay.ToString("F2"));

      double avgBase = machines[2].mean;
      double totBase = avgBase * nPulls;
      Console.WriteLine("\nBaseline average pay = " +
        avgBase.ToString("F2"));
      Console.WriteLine("Total baseline pay   = " +
        totBase.ToString("F2"));

      double regret = totBase - totPay;
      Console.WriteLine("\nTotal regret = " +
        regret.ToString("F2"));

      Console.WriteLine("\nEnd bandit demo \n");
      Console.ReadLine();

    } // Main

    static double ExploreExploit(Machine[] machines,
      double pctExplore, int nPulls)
    {
      // Use basic explore-exploit algorithm
      // Return the average pay per pull
      int nMachines = machines.Length;
      Random r = new Random(2); // which machine

      double[] explorePays = new double[nMachines];
      double totPay = 0.0;

      int nExplore = (int)(nPulls * pctExplore);
      int nExploit = nPulls - nExplore;

      Console.WriteLine("\nStart explore phase");
      for (int pull = 0; pull < nExplore; ++pull)
      {
        int m = r.Next(0, nMachines); // pick a machine
        double pay = machines[m].Pay(); // play
        Console.Write("[" + pull.ToString().PadLeft(3) + "]  ");
        Console.WriteLine("selected machine " + m + ". pay = " +
          pay.ToString("F2").PadLeft(6));
        explorePays[m] += pay; // update
        totPay += pay;
      } // Explore

      int bestMach = BestIdx(explorePays);
      Console.WriteLine("\nBest machine found = " + bestMach);

      Console.WriteLine("\nStart exploit phase");
      for (int pull = 0; pull < nExploit; ++pull)
      {
        double pay = machines[bestMach].Pay();
        Console.Write("[" + pull.ToString().PadLeft(3) + "] ");
        Console.WriteLine("pay = " +
          pay.ToString("F2").PadLeft(6));
        totPay += pay; // accumulate
      } // Exploit

      return totPay / nPulls; // avg payout per pull

    } // ExploreExploit

    static int BestIdx(double[] pays)
    {
      // Index of array with largest value
      int result = 0;
      double maxVal = pays[0];
      for (int i = 0; i < pays.Length; ++i)
      {
        if (pays[i] > maxVal)
        {
          result = i;
          maxVal = pays[i];
        }
      }
      return result;
    }
  } // Program class

  public class Machine
  {
    public double mean; // Avg payout per pull
    public double sd; // Variability about the mean
    private Gaussian g; // Payout generator

    public Machine(double mean, double sd, int seed)
    {
      this.mean = mean;
      this.sd = sd;
      this.g = new Gaussian(mean, sd, seed);
    }

    public double Pay()
    {
      return this.g.Next();
    }

    // -----
    private class Gaussian
    {
      private Random r;
      private double mean;
      private double sd;

      public Gaussian(double mean, double sd, int seed)
      {
        this.r = new Random(seed);
        this.mean = mean;
        this.sd = sd;
      }

      public double Next()
      {
        double u1 = r.NextDouble();
        double u2 = r.NextDouble();
        double left = Math.Cos(2.0 * Math.PI * u1);
        double right = Math.Sqrt(-2.0 * Math.Log(u2));
        double z = left * right;
        return this.mean + (z * this.sd);
      }
    }
    // -----

  } // Machine

} // ns
```

## The Machine Class

The demo program defines class Machine in a very simple way. There are three class fields:

```
public class Machine
{
  public double mean; // Avg payout per pull
  public double sd; // Variability about the mean
  private Gaussian g; // Payout generator
...
```

The Machine class is primarily a wrapper around a Gaussian random number generator. There are many possible design alternatives but in general I prefer to keep my class definitions as simple as possible. Instead of using the standard deviation, as I've done here, some research articles use the mathematical variance. Standard deviation and variance are equivalent because the variance is just the standard deviation squared.

The Machine class has a single constructor that sets up the Gaussian generator:

```
public Machine(double mean, double sd, int seed)
{
  this.mean = mean;
  this.sd = sd;
  this.g = new Gaussian(mean, sd, seed);
}
```

The Machine class has a single public method that returns a Gaussian distributed random payout:

```
public double Pay()
{
  return this.g.Next();
}
```

An alternative to returning a Gaussian distributed payout is to return a uniformly distributed value between specified endpoints. For example, a machine could return a random value between -2.0 and +3.0, where the average payout would be (-2 + 3) / 2 = +0.5 units.

## The Explore-Exploit Implementation

The definition of method ExploreExploit begins with:

```
static double ExploreExploit(Machine[] machines, double pctExplore,
  int nPulls)
{
  int nMachines = machines.Length;
  Random r = new Random(2); // Which machine
  double[] explorePays = new double[nMachines];
  double totPay = 0.0;
...
```

The Random object r is used to select a machine at random during the explore phase. The array named explorePays holds the cumulative payouts for each machine during the explore phase. There's only a need for a single variable, totPay, to hold the cumulative payout of the exploit phase because only a single machine is used.

Next, the number of pulls for the explore and exploit phases is calculated:

```
int nExplore = (int)(nPulls * pctExplore);
int nExploit = nPulls - nExplore;
```

It would be a mistake to calculate the number of exploit pulls using the term (1.0 - pctExplore) because of possible round off in the calculation.

The explore phase, without WriteLine statements, is:

```
for (int pull = 0; pull < nExplore; ++pull)
{
  int m = r.Next(0, nMachines); // Pick a machine
  double pay = machines[m].Pay(); // Play
  explorePays[m] += pay; // Update
  totPay += pay;
}
```

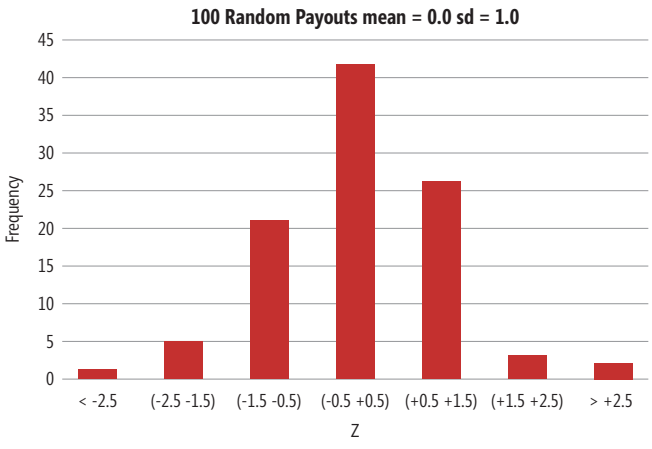



Figure 3 **100 Random Gaussian Values**

The Random.Next(int minVal, int maxVal) returns an integer value between minVal (inclusive) and maxVal (exclusive), so if nMachines = 3, r.Next(0, nMachines) returns a random integer value of 0, 1 or 2.

Next, the best machine found during the explore phase is determined, and used in the exploit phase:

```
int bestMach = BestIdx(explorePays);
for (int pull = 0; pull < nExploit; ++pull)
{
  double pay = machines[bestMach].Pay();
  totPay += pay; // Accumulate
}
```

Program-defined helper method BestIdx returns the index of the cell of its array argument that holds the largest value. There are dozens of variations of the multi-armed bandit problem. For example, some variations define the best machine found during the explore phase in a different way. In my cranky opinion, many of these variations are nothing more than solutions in search of a research problem.

Method ExploreExploit finishes by calculating and returning the average payout per pull over all nPulls plays:

```
  . . .
  return totPay / nPulls;
}
```

Alternative designs could be to return the total payout instead of the average payout, or return the total regret value, or return both the total payout and the average payout values in a two-cell array or as two out-parameters.

## Other Algorithms

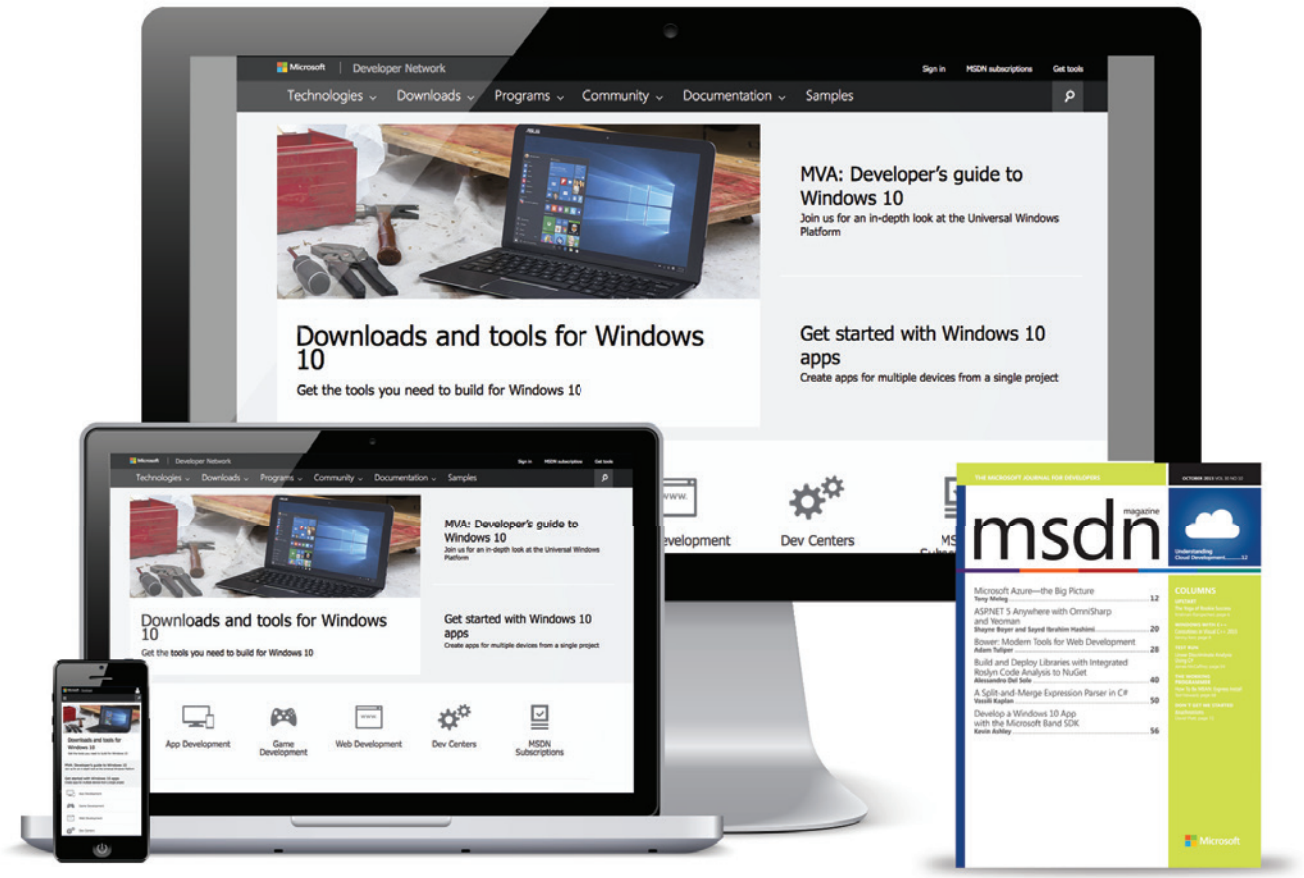
Research suggests that there's no single algorithm that works best for all types of multi-armed bandit problems. Different algorithms have different strengths and weaknesses, depending mostly on the number of machines in the problem, the number of pulls available, and the variability of the payout distribution functions. ■

**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 who reviewed this article: *Miro Dudik and Kirk Olynyk*

# msdn magazine

## Where you need us most.




Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

# CODE Anaheim

## FOR A BETTER TOMORROW

SEPT. 26-29, 2016

HYATT REGENCY
DISNEYLAND®
GOOD NEIGHBOR HOTEL

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

ANAHEIM 2016 CAMPAIGN FOR CODE

**VISUAL STUDIO LIVE!** (VSLive!™) is blazing new trails on the Campaign for Code: For the first time ever, we are headed to Anaheim, CA, to bring our unique brand of unbiased developer training to Southern California. With our host hotel situated just down the street from Disneyland®, developers, software architects, engineers, designers and more can code by day, and visit the Magic Kingdom by night. From Sept. 26-29, explore how the code you write today will create a better tomorrow—not only for your company, but your career, as well!

SUPPORTED BY

Microsoft msdn magazine Visual Studio MAGAZINE Visual Studio

PRODUCED BY

1105 MEDIA
YOUR GROWTH, OUR BUSINESS.

# ANAHEIM • SEPT. 26-29, 2016

## HYATT REGENCY, A DISNEYLAND® GOOD NEIGHBOR HOTEL

### CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

### DEVELOPMENT TOPICS INCLUDE:

- Windows Client
- Visual Studio/.NET
- Windows Client
- Mobile Client
- JavaScript/HTML5 Client
- ASP.NET
- Cloud Computing
- Database and Analytics

## California code with us: register today!

REGISTER NOW AND SAVE $300!

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

USE PROMO CODE VSLAN2

VSLIVE.COM/ANAHEIM

# VOTE "YES" FOR BETTER CODE

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

## Washington, D.C.

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

WASHINGTON D.C. 2016 CAMPAIGN FOR CODE

**VISUAL STUDIO LIVE!** (VSLive!™) is on a Campaign for Code in 2016, in support of developer education. It's only fitting that we return with our unique brand of practical, unbiased, Developer training to the nation's capital this year. From Oct. 3- 6, we're offering four days of sessions, workshops and networking events to developers, software architects, engineers and designers—all designed to help you vote "yes" for better code and write winning applications across all platforms.

SUPPORTED BY

Microsoft msdn magazine Visual Studio MAGAZINE Visual Studio

PRODUCED BY

1105 MEDIA YOUR GROWTH. OUR BUSINESS.

# WASHINGTON, D.C. • OCT. 3-6, 2016

RENAISSANCE, WASHINGTON, D.C.


## CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

### DEVELOPMENT TOPICS INCLUDE:

- Visual Studio/.NET
- Windows Client
- Mobile Client
- JavaScript/HTML5 Client
- ASP.NET
- Cloud Computing
- Database and Analytics

## Do your developer duty: register to join us today

REGISTER NOW AND SAVE $300!

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

USE PROMO CODE VSLDC2

VSLIVE.COM/DC



TED NEWARD

# How To Be MEAN: Getting the Edge(.js)

Welcome back, "MEANers." In the previous installment, I added a bit of structure to the otherwise structureless environment that is JavaScript, Node and MongoDB, by adding the MongooseJS library to the software stack I've slowly been building. This put some "schema" around the different collections that the Node/Express middleware was receiving and storing, which is nice because it helps avoid some common human-inspired errors (such as searching for "fristName" instead of the actual field "firstName"). Best of all, MongooseJS is entirely code-side, which means that for all practical purposes, you now have the best of both worlds, at least as far as the database is concerned—"schemaless" in the database (making it far easier to refactor) and "schemaful" in the code (making it far less likely that a typo will screw things up).

The key here is to realize that the body of the function—whether written in C# or in F#—is of a specific .NET-type signature: Func<object, Task<object>>.

But, if I can take a personal moment here, I must admit that I miss the Microsoft .NET Framework. Or, to be more specific, I miss some of the very cool things that the .NET ecosystem has available within it. Particularly, when I'm executing on the Microsoft Azure cloud, where a number of organizations are going to have some small (or very large) investment in the .NET "stack," it seems a little out of place to be talking so much about JavaScript, if all of that .NET stuff remains out of reach. Or, at least, out of reach except for doing some kind of long-haul HTTP-style request, which seems kind of silly when you're operating inside the same datacenter.

Fortunately, we have an edge. Or, to be more specific, Edge.js.

## Edge.js

The Edge.js project is seriously one-of-a-kind in a lot of ways, most notably that it seeks to very directly address the "platform gap" between .NET and Node.js. Hosted at bit.ly/1W7xJmo, Edge.js deliberately seeks to make each platform available to the other in a very code-friendly way to each.

For example, getting a Node.js code sample to call a .NET function looks like this:

```
var edge = require('edge');

var helloWorld = edge.func(function () {/*
  async (input) => {
    return ".NET Welcomes " + input.ToString();
  }
*/});

helloWorld('JavaScript', function (error, result) {
  if (error) throw error;
  console.log(result);
});
```

As you can see, programmatically, this isn't difficult: Pass a function literal to the edge.func method and inside that function literal include the .NET code to invoke as the body of a comment.

Yes, dear reader, a comment. This isn't so strange when you realize:

- It can't be literal C# syntax or the Node interpreter wouldn't recognize it as legitimate program syntax (because, after all, the Node interpreter is a JavaScript interpreter, not a C# interpreter).
- Unlike a compiled program, the interpreter has access to the full body of whatever source code is defined there, rather than just what the compiler chose to emit.

Note that this isn't limited to just C#, by the way; the Edge.js project lists several other languages that can be used as the "target" of an Edge call, including F#, Windows PowerShell, Python or even Lisp, using .NET implementations of each of those languages. My favorite, of course, is F#:

```
var edge = require('edge');

var helloFs = edge.func('fs', function () {/*
  fun input -> async {
    return "F# welcomes " + input.ToString()
  }
*/});

helloFs('Node.js', function (error, result) {
  if (error) throw error;
  console.log(result);
});
```

Note the key difference is an argument slipped in front of the function literal, indicating which language is being passed in that function comment.

The key here is to realize that the body of the function—whether written in C# or in F#—is of a specific .NET-type signature: Func<object, Task<object>>. The asynchrony here is necessary because remember that Node prefers callbacks to direct sequential execution in order to avoid blocking the main Node.js event loop.

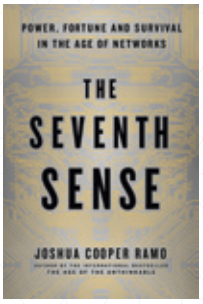
Edge.js also makes it relatively easy to invoke these functions on compiled .NET DLLs. So, for example, if you have a compiled

# ACQUIRE
Conference & EXPO

JUNE 8-9 2016
WASHINGTON, DC
WALTER E. WASHINGTON CONVENTION CENTER

ACQUIREshow.com

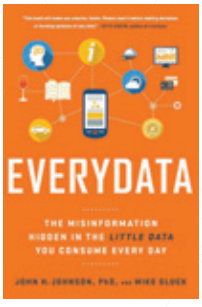
## Meet these renowned authors—only at ACQUIRE!




**OPENING KEYNOTE:**
Joshua Cooper Ramo

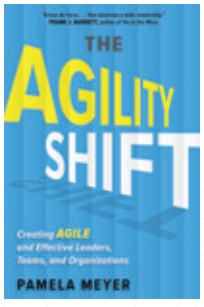

David L. Hall


Pagan Kennedy

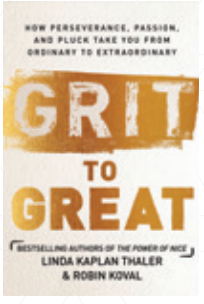



John H. Johnson, Ph.D.


William D. Eggers




Pamela Meyer




Linda Kaplan Thaler

**PLUS:** Keynote address by Mark Kelly & Gabby Giffords • 100+ training sessions across 7 tracks • 200+ exhibitors • Happy Fed Pavilion for the federal consumer • and much more!

**FREE FOR GOVERNMENT AND MILITARY!**

# REGISTER NOW!
## ACQUIREshow.com

AN FCW EVENT

FOUNDING PARTNER PSC PROFESSIONAL SERVICES COUNCIL

assembly of .NET code that wants to be invoked, Edge.js can invoke it so long as the assembly name, type name and method name are provided as part of the "func" call:

```
var helloDll = edge.func({
  assemblyFile: "Echo.dll",
  typeName: "Example.Greetings",
  methodName: "Greet"
});
```

If the type signature of Greet is a Func<object, Task<object>>, such as the one shown in **Figure 1**, then Node.js can call it using the same invocation pattern (passing in input arguments and a function callback), as shown for the other examples.

It's also possible to go the other way—to arrange .NET code to call into Node.js packages—but because the goal here is to work Node.js on the server side, I'll leave that as an exercise to the interested reader. (By the way, all the Edge.js stuff is much easier to work with from a Windows machine than a Mac; trying to get it to work on my Mac during the writing of this column was definitely too time-consuming, all things considered, so this is one case where the Windows experience definitely trumps the Mac for Node.js-related development.)

I want to give this a quick hello-world-style spin before doing anything more complicated.

## Hello, Edge

First things first, again, all of this needs to work in Microsoft Azure (because that's my chosen target deployment environment), so make sure to "npm install --save edge" so that it'll be tracked in the package.json manifest when it gets committed to Azure. Next, I add the "helloWorld" function to the app.js code, and set up a quick endpoint so that I can "GET" on it and get that greeting back via HTTP, as shown in **Figure 2**. And, sure enough, sending a GET to msdn-mean.azurewebsites.net/edgehello brings back:

```
{"message":".NET Welcomes Node, JavaScript, and Express"}
```

## Hello, SQL Server

One question that periodically arises whenever I talk about the MEAN stack with .NET developers is about MongoDB. A fair number of people don't like the idea of giving up their SQL Server, particularly when running on Azure. Of course, the Node.js community has built several relational database access APIs and SQL Server is just a TDS connection away from any of those, but Edge.js actually has a pretty fascinating solution to that particular problem (once you "npm install --save edge-sql," to pull in the Edge-SQL package):

```
var edge = require('edge');

var getTop10Products = edge.func('sql', function () {/*
  select top 10 * from Products
*/});

getTop10Products(null, function (error, result) {
  if (error) throw error;
  console.log(result);
  console.log(result[0].ProductName);
  console.log(result[1].ReorderLevel);
});
```

This code is assuming that the Azure environment has an environment variable called EDGE_SQL_CONNECTION_STRING set to the appropriate SQL Server connection string (which, in this case, would presumably point to the SQL Server instance running in Azure).

That's arguably simpler than working with just about anything else I've seen, to be honest. It probably won't replace Entity Framework any time soon, granted, but for quick access to a SQL Server instance, perhaps as part of a "polypraeclusio" ("multiple storage") approach that uses SQL Server for storing the strictly schemaed relational data and MongoDB for the schemaless JSON-ish data, it's actually pretty elegant.

## Why, Again?

Given that most developers generally look askance at any solution that requires them to be conversant in multiple languages at the same time, it's probably worth digging a little deeper into when and how this might be used.

The obvious answer is that for most greenfield kinds of projects, with no legacy code support required, the general rule would be to stay entirely inside of one language/platform or the other: either stick with .NET and use Web API and so on, or go "whole hog" into Node.js, and rely on the various libraries and packages found there to accomplish the goals at hand. After all, for just about anything you can think of doing in the .NET world, there's likely an equivalent package in the npm package repository. However, this approach comes with a few caveats.

First, the .NET ecosystem has the advantage of having been around a lot longer and, thus, some of the packages there being more battle-tested and trustworthy. Many of the npm packages are still flirting with highly dubious version numbers; managers have a hard time trusting anything with a verison number starting with 0, for example.

Figure 1 **An Edge.js-Compatible .NET Endpoint**

```
using System;
using System.Threading.Tasks;

namespace Example
{
  public class Greetings
  {
    public async Task<object> Greet(object input)
    {
      string message = (string)input;
      return String.Format("On {0}, you said {1}",
        System.DateTime.Now,
        Message);
    }
  }
}
```

Figure 2 **Adding the "helloWorld" Function**

```
var helloWorld = edge.func(function () {/*
    async (input) => {
        return ".NET Welcomes " + input.ToString();
    }
*/});

var edgehello = function(req, res) {
  helloWorld('Node, JavaScript, and Express', function (err, result) {
    if (err) res.status(500).jsonp(err);
    else res.status(200).jsonp( { message: result } );
  });
};

// ...

app.get('/edgehello', edgehello);
```

Second, certain problems lend themselves better to certain programming approaches or environments; case in point, the F# language frequently "makes more sense" to those with a mathematical background and makes it easier sometimes for them to write certain kinds of code. Such was the case a few years ago when I wrote the Feliza library (as part of the series demonstrating how to use SMS in the cloud using the Tropo cloud); while it could have been written in C#, F#'s pattern-matching and "active patterns" made it much easier to write than had I done it in C#. The same is true for JavaScript (perhaps more so).

Last, and perhaps most important, is that sometimes the environment in which the code is executing just naturally favors one platform over another. For example, numerous organizations that host applications in Azure use Active Directory as their authentication and authorization base; as a result, they'll want to continue to use Active Directory for any new applications written, whether in .NET or Node.js. Accessing Active Directory is generally much simpler and easier to do from a .NET environment than anything else, so the Edge.js library offers a convenient "trap door," so to speak, to make it much easier to access Active Directory.

## Wrapping Up

It's been a little lighter this time around, largely because the Edge.js library takes so much of the work out of working interoperably with the .NET environment. And it opens a whole host of new options for the Azure MEANer, because now you have access to not just one but two rich, full ecosystems of tools, libraries and packages.

Note that there's a whole column waiting to be written going the other direction, too, by the way: In many cases, certain kinds of applications are much easier to write using the various packages available in the npm repository, that Edge.js now opens up to the traditional .NET developer. And with the recent open source release of Chakra—the Microsoft JavaScript core that can "plug in" to the Node environment as a drop-in plug-in—this opens up even more opportunities to use JavaScript as part of a "standard" .NET application, including even the opportunity to host a JavaScript interpreter at the center of your application. This has its own interesting implications when you stop to think about it.

There are a few more things I want to discuss before moving off the server completely, but I'm out of space, so for now … happy coding! ■

**Ted Neward** *is a Seattle-based polytechnology consultant, speaker and mentor. He has written more than 100 articles, is an F# MVP, INETA speaker, and has authored and coauthored a dozen books. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

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


Create **high-quality software**
always aligned with your business!

CodeFluent Entities is a product integrated into Visual Studio that creates your application foundations.

Minimize heavy plumbing work and internal framework development by generating your data access layers.

Keep your business in sync with your software implementations and focus on what makes the difference.

**MODEL YOUR BUSINESS**
Leverage the graphical modeler to define your business model in a centralized place

**DEFINE YOUR FOUNDATIONS**
Generate a direct translation of your model into ready-to-use data and business layers

**DEVELOP YOUR APPLICATIONS**
Build modern applications by focusing on your high-value custom code

softfluent

Visual Studio Partner

Innovative software company founded in 2005 by Microsoft veterans

SoftFluent - 2018 156th Avenue NE Bellevue, WA 98007 USA - info@softfluent.com - www.softfluent.com

Copyright © 2005 - 2015, SoftFluent SAS. All rights reserved.
Microsoft, Visual Studio and Excel® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.



# Left Brains for the Right Stuff

I've been reading Apollo astronaut memoirs lately. The very best is Michael Collins' "Carrying the Fire" (Rowman & Littlefield, 2001) in which he vents his frustrations on the guidance computer in the command module simulator:

> *I lost my temper. Flash an "operator error" light at me, will you, you stupid goddamned computer, and I would sputter and stammer until the soothing voice of [the simulation operator] came over the earphones and unctuously explained how I had offended their precocious brat.*

Intrigued, I decided to dig deeper into those earliest spacecraft computers. That meant looking up an old comrade-in-arms. Hugh Blair-Smith hired me in 1987, for a startup that imploded two years later (not his fault). He was also my student in 1992, in my first Windows class at Harvard Extension (16-bit SDK in C, because you ask). But long before all that, he had worked at MIT's Instrumentation (later Draper) Lab, helping to develop both hardware and software for the Apollo Guidance Computer. The same machine, with different software, was used in both the Command Module and the Lunar Module (bit.ly/1RDrUj4) in its more advanced "Block II" configuration.

I hadn't talked with Blair-Smith since writing "Why Software Sucks" 10 years ago (he's in it, but not by name). Hugh has now written a book about his Apollo days, entitled "Left Brains for the Right Stuff: Computers, Space, and History" (SDP Publishing, 2015). He describes the hardware, such as the rope memory that held the ferrite core ROM (bit.ly/1pBCtGk), and the software, down to the microcode of the divide instruction (and in a geeky endnote, even the nanocode of the multiply instruction).

Not only does Blair-Smith's book cover technical topics, he also discusses the social and political context of the space race within which all this engineering took place. And he does so in beautiful language. Here he recounts watching Apollo 8's TV broadcast on Christmas Eve 1968, from the first humans to orbit the moon:

> *The spacecraft coasted on toward the darkness that only the shadow of an airless world can produce. "And now, from the crew of Apollo 8, we close with, good night, good luck, a Merry Christmas, and God bless all of you, all of you on the good Earth." And to you, Frank and Jim and Bill, I thought. And to you, little computer.*

I found the UI especially fascinating. The astronauts entered verbs and nouns through a numerical keypad. For example, "Verb 88, enter," told the computer to ignore input from the VHF rangefinder, while "Verb 87, enter" told it to start paying attention again.

"This was only intended as a stopgap until they could think of something better," says Blair-Smith. "But it turns out the astronauts sort of liked it. 'Even a pilot can use this,' Dave Scott (Apollo 15 commander) said, so it stuck." (See Scott's article, "The Apollo Guidance Computer: A User's View," downloadable as a PDF at bit.ly/1pLV5V5.)

The computer would prompt the astronauts to take actions, but couldn't initiate a rocket engine burn on its own. Blair-Smith says, "The philosophy is that no significant action can be taken without an overt command from the crew—the astronauts insisted that, 'If we kill ourselves, it will be our fault, not some goddamn computer's.'"

Naturally, I wanted to hear more. Blair-Smith was kind enough to invite me to the quarterly lunch of the surviving Apollo engineers, in Cambridge near MIT.

Not only does Hugh's book cover technical topics, he also discusses the social and political context of the space race within which all this engineering took place.

Of course I had to bring my daughter Annabelle, a budding space geek. For her imminent 16th birthday (and how the heck did that happen?), she wants a Buzz Aldrin T-shirt that says, "Get Your Ass to Mars." The old men loved meeting a fresh newcomer to pass the torch. One said, "I'm the Rope Mother for Apollo 12," (the single engineer with ultimate responsibility for the ROM), with pride that hasn't dimmed, and won't while the gentleman lives.

Annabelle and I saw the full moon rising over the salt marshes near our home a few nights later. Bright, smiling, mithril-colored, but oh so lonely, with no footprints for going on 50 years. "We could do that, you know," Annabelle said to me. She'd caught the spark.

I wondered what the Apollo veterans see when they look at the moon. As always, Blair-Smith sums it up pithily: "Notice something? For us who lived Apollo, everything about it remains in present tense, and always will."

With one exception: a phrase I heard them use to each other, to me and Annabelle, and to themselves. I now pass those words to you, dear reader, as their challenge to us all, and as their epitaph when they come to need one: "We didn't know what we couldn't do. So we just went ahead and did it." ■

**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.*

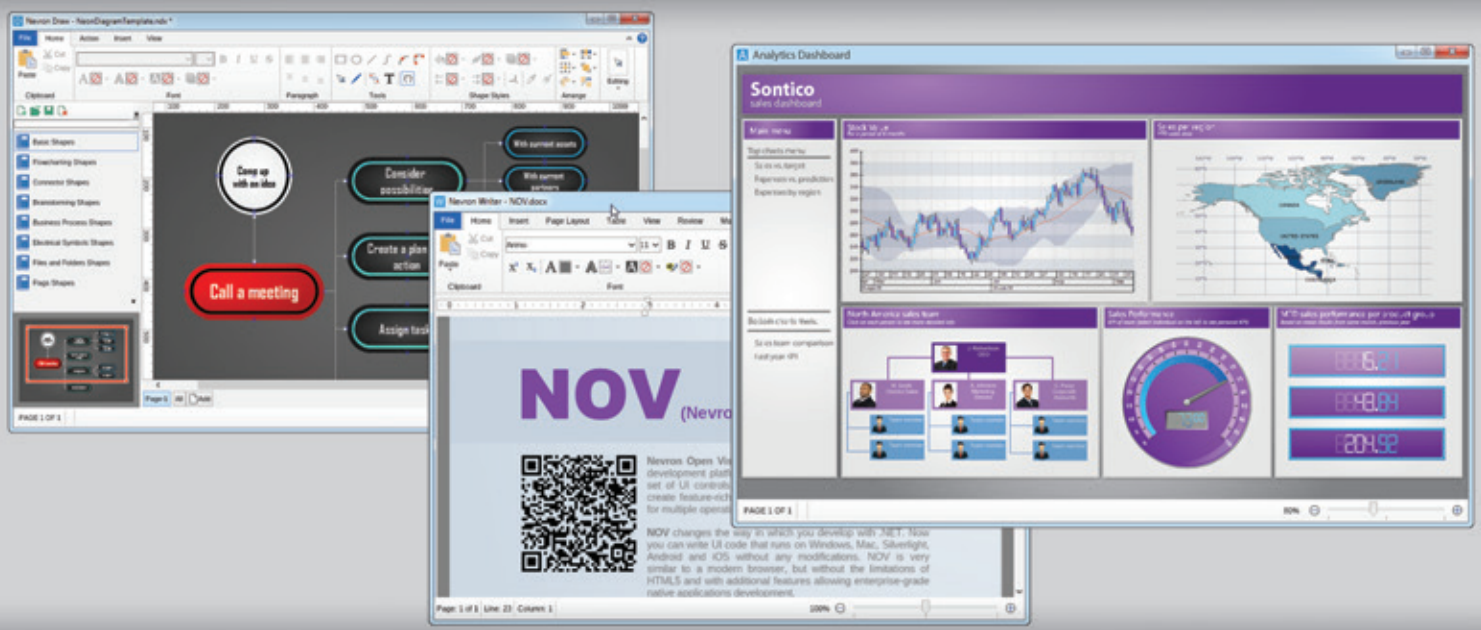
NEVRON

www.nevron.com

NEVRON
OPEN VISION for .NET

# The Complete UI suite for .NET

| free | | | | new | new | new | new | soon |
|---|---|---|---|---|---|---|---|---|
| UI | RICH TEXT | BARCODE | GAUGE | DIAGRAM | MAP | CHART | SCHEDULER | GRID |




Nevron Open Vision is the first and only .NET component suite that allows you to develop cutting edge Applications, Dashboards and User Interfaces for Windows (WinForms and WPF) and Mac (MonoMac and Xamarin.Mac) from a Single Code Base.

It includes advanced UI controls such as Chart, Diagram, Grid, Text Editor, Scheduler, Gauges and Barcodes as well a complete set of UI controls (ListBox, TreeView, ComboBox, Color Pickers etc.).

The controls in those suites seamlessly integrate in

Microsoft Silverlight | WPF | WinForms | Xamarin

All controls in the suite are completely windowless, styleable and open for customization.

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

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

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

Visual Studio Partner

# SYNCFUSION *SUCCINCTLY* SERIES


- 90 titles and growing
- Ad-free
- 100 pages
- PDF and Kindle formats

syncfusion.com/msdnroslyn

Syncfusion®