

msdn magazine



Microsoft
Big Data Tools.....20, 28, 36

Your Next Great Dashboard Starts Here

Create high impact and information rich decision support systems for desktops and the web with the DevExpress Universal Subscription.

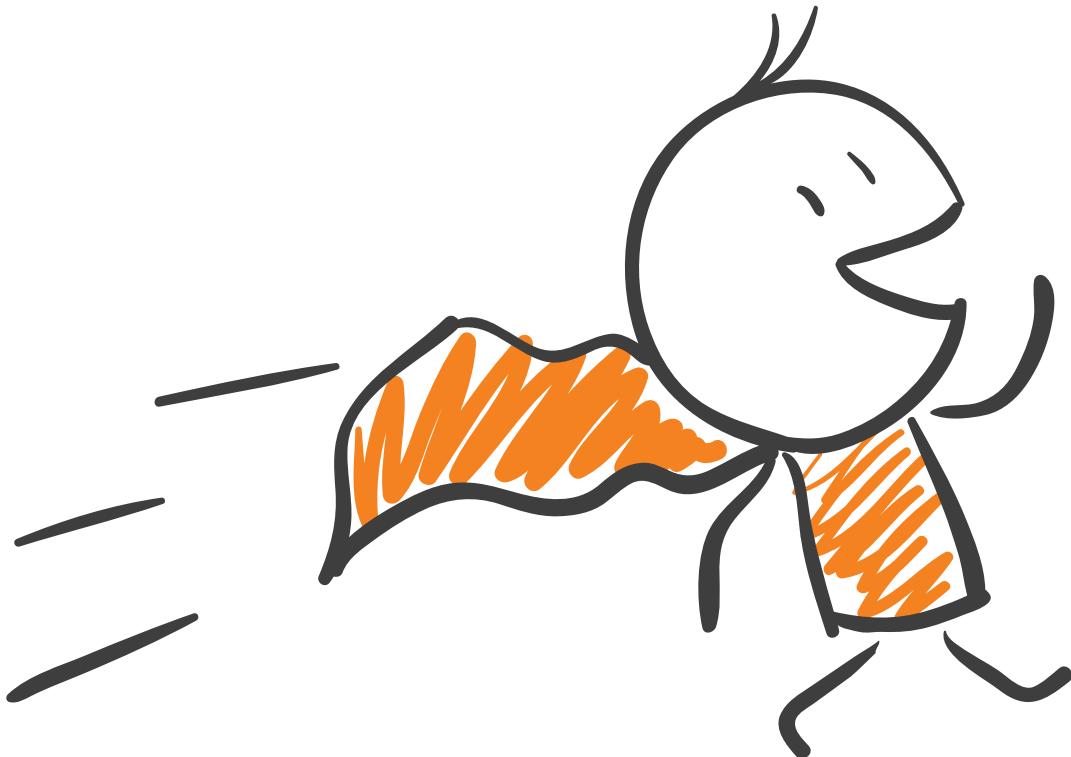


Download Your Free 30-Day Trial

devexpress.com/dashboard

Unleash the UI Superhero in You

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



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

msdn magazine



Microsoft
Big Data Tools.....20, 28, 36

Making Big Data Batch Analytics Easier Using U-SQL Michael Rys	20
Real-Time Data Analytics for .NET Developers Using HDInsight Omid Afnan	28
Creating Big Data Pipelines Using Azure Data Lake and Azure Data Factory Gaurav Malhotra	36
Using the OneDrive REST API in a Windows 10 App Laurent Bugnion	44
Babylon.js: Advanced Features for Enhancing Your First Web Game Raanan Weber	54

COLUMNS

UPSTART

A Winning Plan
Krishnan Rangachari, page 6

CUTTING EDGE

Don't Gamble with UX—
Use Wireframes
Dino Esposito, page 8

DATA POINTS

EF7 Migrations: Not New but Definitely Improved
Julie Lerman, page 14

THE WORKING PROGRAMMER

How To Be MEAN:
Test Me MEANly
Ted Neward, 66

ESSENTIAL .NET

C# Scripting
Mark Michaelis, page 70

DON'T GET ME STARTED

Moving Forward, Looking Back
David S. Platt, page 80

ASP.NET MVC!

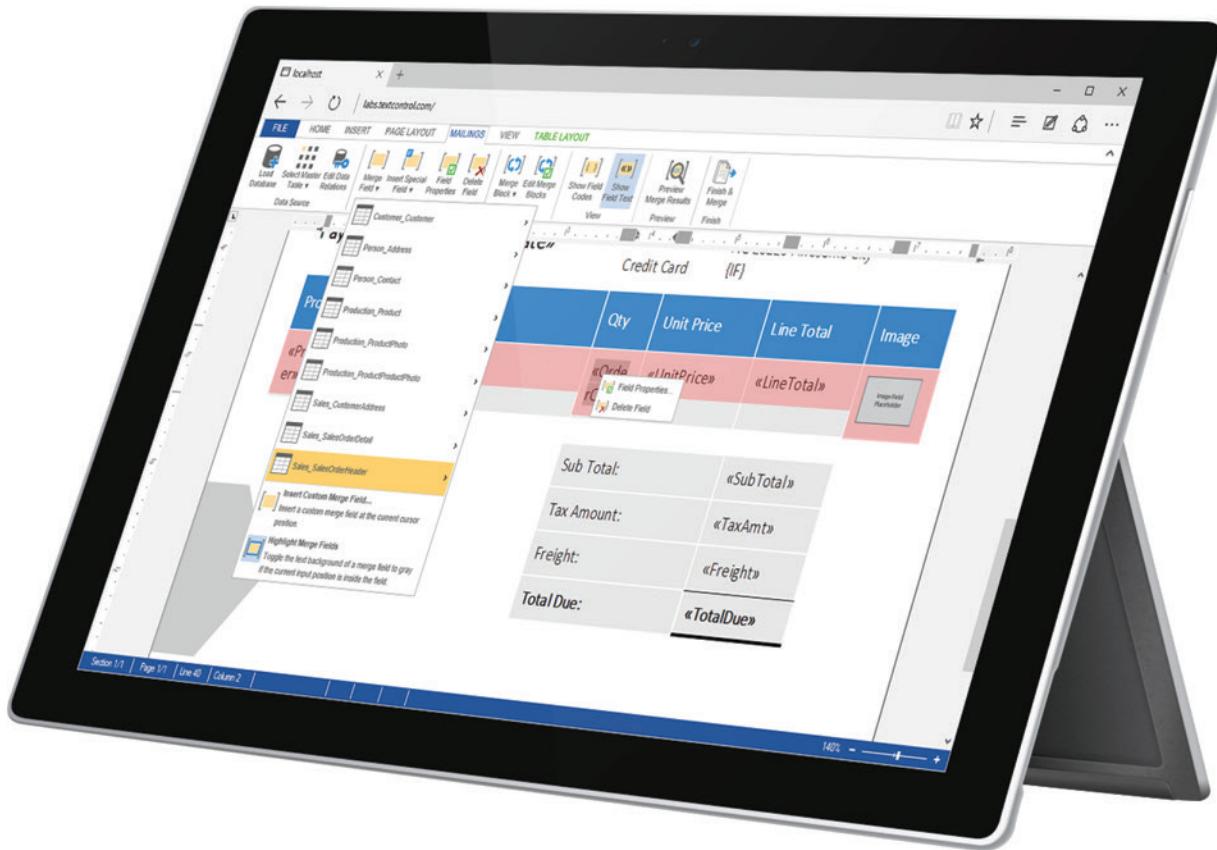
The first cross-browser,
true WYSIWYG
HTML5-based
document editor.

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.



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



TEXT CONTROL

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, Julie Lerman, Mark MichaelisTed Neward,

David S. Platt, Bruno Terkaly, Ricardo Villalobos

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 Cvjanovic

Graphic Designer Erin Horlacher

Senior Graphic Designer Alan Tao

Senior Web Designer Martin Peace

PRODUCTION STAFF

Director, Print Production David Seymour

Print Production Coordinator Anna Lyn Bayaua

ADVERTISING AND SALES

Chief Revenue Officer Dan LaBianca

Regional Sales Manager Christopher Kourtoglou

Account Executive Caroline Stover

Advertising Sales Associate Tanya Egenolf

ONLINE/DIGITAL MEDIA

Vice President, Digital Strategy Becky Nagel

Senior Site Producer, News Kurt Mackie

Senior Site Producer Gladys Rama

Site Producer Chris Paoli

Site Producer, News David Ramel

Director, Site Administration Shane Lee

Site Administrator Biswarup Bhattacharjee

Front-End Developer Anya Smolinski

Junior Front-End Developer Casey Rysavy

Executive Producer, New Media Michael Domingo

Office Manager & Site Assoc. James Bowling

LEAD SERVICES

Vice President, Lead Services Michele Imgrund

Senior Director, Audience Development & Data Procurement Annette Levee

Director, Audience Development & Lead Generation Marketing Irene Fincher

Director, Client Services & Webinar Production Tracy Cook

Director, Lead Generation Marketing Eric Yoshizuru

Director, Custom Assets & Client Services

Mallory Bundy

Editorial Director, Custom Content Lee Pender

Senior Program Manager, Client Services & Webinar Production Chris Flack

Project Manager, Lead Generation Marketing Mahal Ramos

Coordinator, Lead Generation Marketing

Obum Ukabam

MARKETING

Chief Marketing Officer Carmel McDonagh

Vice President, Marketing Emily Jacobs

Senior Manager, Marketing Christopher Morales

Marketing Coordinator Alicia Chew

Marketing & Editorial Assistant Dana Friedman

ENTERPRISE COMPUTING GROUP EVENTS

Senior Director, Events Brent Sutton

Senior Director, Operations Sara Ross

Director, Event Marketing Merikay Marzoni

Events Sponsorship Sales Danna Vedder

Senior Manager, Events Danielle Potts

Coordinator, Event Marketing Michelle Cheng

Coordinator, Event Marketing Chantelle Wallace

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

Chief Executive Officer

Rajeev Kapur

Chief Operating Officer

Henry Allain

Vice President & Chief Financial Officer

Michael Rafter

Executive Vice President

Michael J. Valenti

Chief Technology Officer

Erik A. Lindgren

Chairman of the Board

Jeffrey S. Klein

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

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

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

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

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

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

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

Reaching the Staff

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

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

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

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

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

 Microsoft

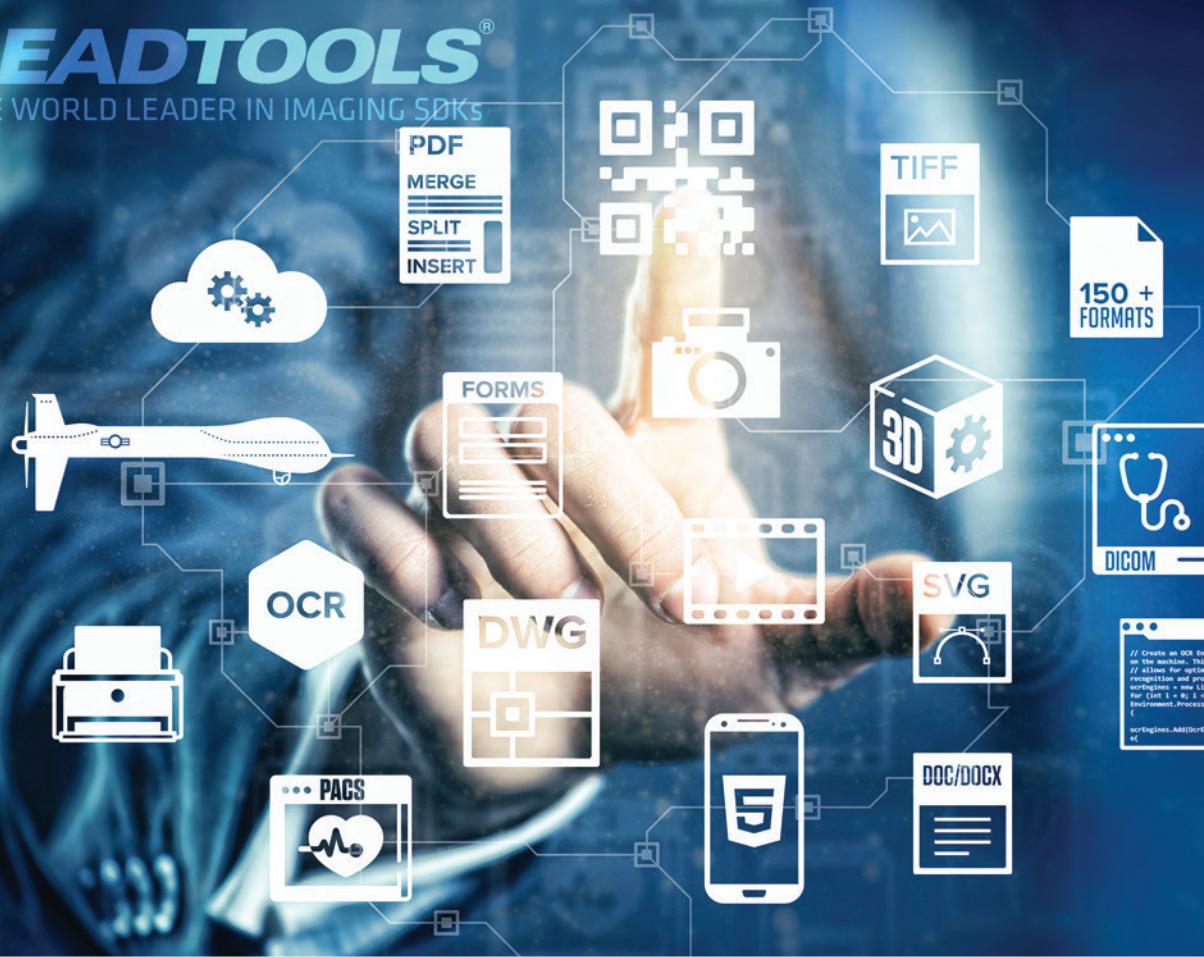

BPA
WORLDWIDE


AMERICAN BUSINESS MEDIA
A Century Moving Business Forward


WPA
Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs



MILLIONS OF LINES OF CODE AT YOUR FINGERTIPS

Document Imaging

Document Viewer

Document Converter

OCR, MICR, OMR & ICR

Barcode & Forms Recognition

Create, Save, Edit & View PDF

Annotations, TWAIN & SVG

Medical Imaging

DICOM, CCOW & HL7

PACS Client & Server Framework

DICOM Storage Server Framework

Web & Desktop Viewers

Image Processing & Annotations

Medical 3D (MPR, MIP, VRT)

Multimedia Imaging

Play, Capture, Convert & DVR

Stream - MPEG2-TS & RTSP

Media Streaming Server

MPEG-4, H.264 & H.265

Media Foundation & DirectShow

Distributed Transcoding

.NET

Windows API

WinRT

Linux

iOS

OS X

Android

HTML5

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD
TECHNOLOGIES
INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840





EDITOR'S NOTE

MICHAEL DESMOND

Go Big or Go Home

Big Data is fast becoming big business, and producing a lot of development activity as a result. Market forecasts from research outfit Wikibon project that spending on Big Data technology and services will reach \$49.28 billion in 2018, up from \$27.36 billion in 2014.

No surprise, Microsoft is working to empower developers in Big Data scenarios. At the Build conference in San Francisco last April, the company revealed services such as Azure Data Warehouse, Azure Data Lake, and Elastic Databases for Azure SQL Database. As we move into 2016, those efforts are beginning to bear fruit. Which explains why this month's issue of *MSDN Magazine* is focused on Big Data technologies and development.

"The ideal starting point would be to find a way to start collecting the raw data of interest into a Big Data store, and then start to explore datasets, find insights and eventually operationalize the process."

Omid Afnan, Principal Program Manager, Microsoft Azure Big Data Team

Michael Rys, in his feature, "Making Big Data Batch Analytics Easier Using U-SQL," explores the new U-SQL language, which combines SQL and C# semantics to support Big Data scenarios. Next, Omid Afnan shows how to collect, analyze and act on continuous streams of data in real time, in his article, "Real-Time Data Analytics for .NET Developers Using HDInsight." The last article

in the set, Gaurav Malhotra's "Creating Big Data Pipelines Using Azure Data Lake and Azure Data Factory," walks through building a Big Data pipeline using Azure Data Factory to move Web log data to Azure Data Lake Store, and processing that data using a U-SQL script on the Azure Data Lake Analytics service.

Why Big Data and why now? As Afnan notes, 2015 was a big year for Big Data at Microsoft.

"While we have grown internally to multi-exabyte scale in our Big Data scenarios, we have shipped a lot of new capabilities in Azure," Afnan says. "Advances in HDInsight and the release of Azure Data Lake make Big Data significantly easier for developers. ADL Analytics and Storage let developers focus on the queries and aggregations at the business level by abstracting away the complexity of the distributed computing clusters and map-reduce architecture underneath."

He also cites recent investments in Visual Studio tooling to streamline coding and debugging for ADL and HDInsight. Still, Afnan says dev shops face a challenge as they encounter the steep learning curve around the "three V's" of Big Data—velocity, variety and volume—and the distributed, multi-stage processing models used in the space.

"The ideal starting point would be to find a way to start collecting the raw data of interest into a Big Data store, and then start to explore datasets, find insights and eventually operationalize the process," he says. "Reducing the ramp-up time can be achieved by reducing the number of things you have to manage. Going with a Big Data platform that has a job abstraction is certainly a good way to avoid floundering and giving up."

Microsoft aims to help dev shops engaged in the challenge, delivering tools and platforms that both expose and streamline new capabilities. These include, says Afnan, new developer tools for "more complex debug scenarios for individual node/vertex failures, exploration of data skew issues, and analysis of multiple runs of the same script."

There's a lot to be excited about in the Big Data space. I expect we'll be seeing plenty more about Big Data in our pages over the months and years to come.

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

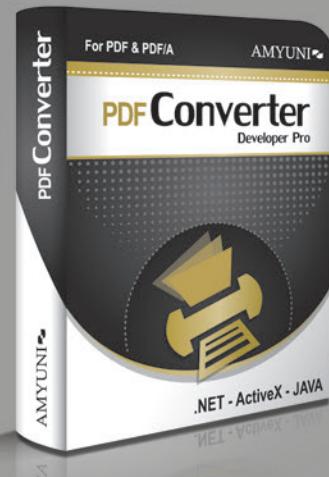
NEW
v5.5

- 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

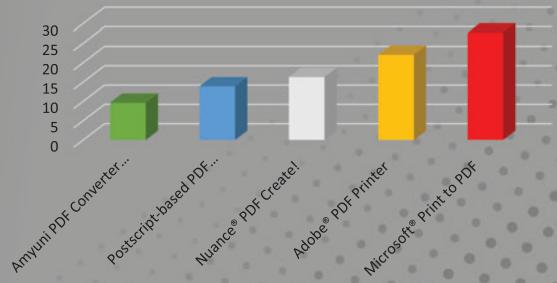
- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML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



High Performance PDF Printer for Desktops and Servers

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

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



Other Developer Components from Amyuni®

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



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

All development tools available at
www.amyuni.com

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

A Winning Plan

As a developer, I struggled to be consistently productive. When work was exciting with obscure bug fixes or high-impact features, my excitement would provide a burst of energy for weeks. At other times, my desire to contribute would conflict with my inertia and unproductivity. Over the years, I've discovered a set of solutions that helped me uncover my "inner beast" at work and help me be reliably effective.

Inertia in the Face of Complexity

Inertia was a challenge I grappled with every week, particularly when I was asked to lead a project. I'd look busy, wait for a change of plans or deadlines, hope for a rescuer, and reply with increasingly clever versions of "I'm working on it." I'd finally start the project a few days before the deadline, having exhausted all my energy in spinning wheels for weeks prior. While I'd often get recognized for great work, I grew tired with the sheer inefficiency of it all.

The best way for me to deal with inertia is to acknowledge it, smile at it and finally ignore it by doing the work anyway.

Today, instead, I just start work on some part, *any part* of the project; the path forward soon becomes clear.

Every day, I do the *one* task that can move the project forward with maximum impact. I do this by writing down on paper—the previous night—the singular, smallest, highest-impact action item for each of my top projects. This focuses my mind and eliminates distractions. And by doing this the day prior, I don't fall into the trap of being swayed by the to-and-fro of everyday fire-drills.

I still feel overwhelmed at the beginning of every project, but I accept that overwhelmed feeling as normal and human. Instead of using it as an excuse for inaction, I use it as motivation for taking the tiniest action I can. The best way for me to deal with inertia is to acknowledge it, smile at it and finally ignore it by doing the work anyway.

Strategic Goal-setting

I also began to write down my top-five work goals on paper. I made them simple enough that a child could tell me if I achieved my aims or not, and I set a timeline for each one. For example, I'd write statements like, "I'm promoted to the senior level in my current position by March 31," or, "I fix 10 Pril bugs in feature area X by April 15."

When my goals were vague, I'd trap myself in a gray zone of unsureness and give myself the benefit of the doubt. I'd discover months later that I hadn't lived up to my expectations for the quarter or even my whole career.

Once I wrote my goals down, I began to review them every day, as soon as I woke up. I gradually discovered that I was off-track on almost all my objectives; I could fool others at Scrum meetings, but I couldn't let *myself* down every morning. Soon, I found myself cutting non-essential meetings and becoming more aware of unproductive habits.

I began to look at my life through the lens of my aspirations. The universe—my colleagues, managers, friends, complete strangers—began to present opportunities for me to hit my targets. Of course, such opportunities had always been there—I simply hadn't noticed in the haze of a goal-less life.

Through trial-and-error, I also learned that I had to create a new objective when I was on the cusp of achieving an existing one. For example, if I'm about to achieve my promotion before March 31, I craft a new target—"I'm promoted to the principal level by Dec. 31 of next year." If I didn't proactively replace goals, my mind would gloat in self-congratulation for weeks.

Focusing on Wins

I found that goals and an action-oriented style by themselves weren't enough. My mind still tried to dwell on my weaknesses. To combat this, at the end of every night, I began to track my five daily "wins"—successes that I otherwise may not have acknowledged. If I had a pressing problem area (for example, fixing my relationship with my manager), I'd make it a required task for me to jot down one success for that goal every night.

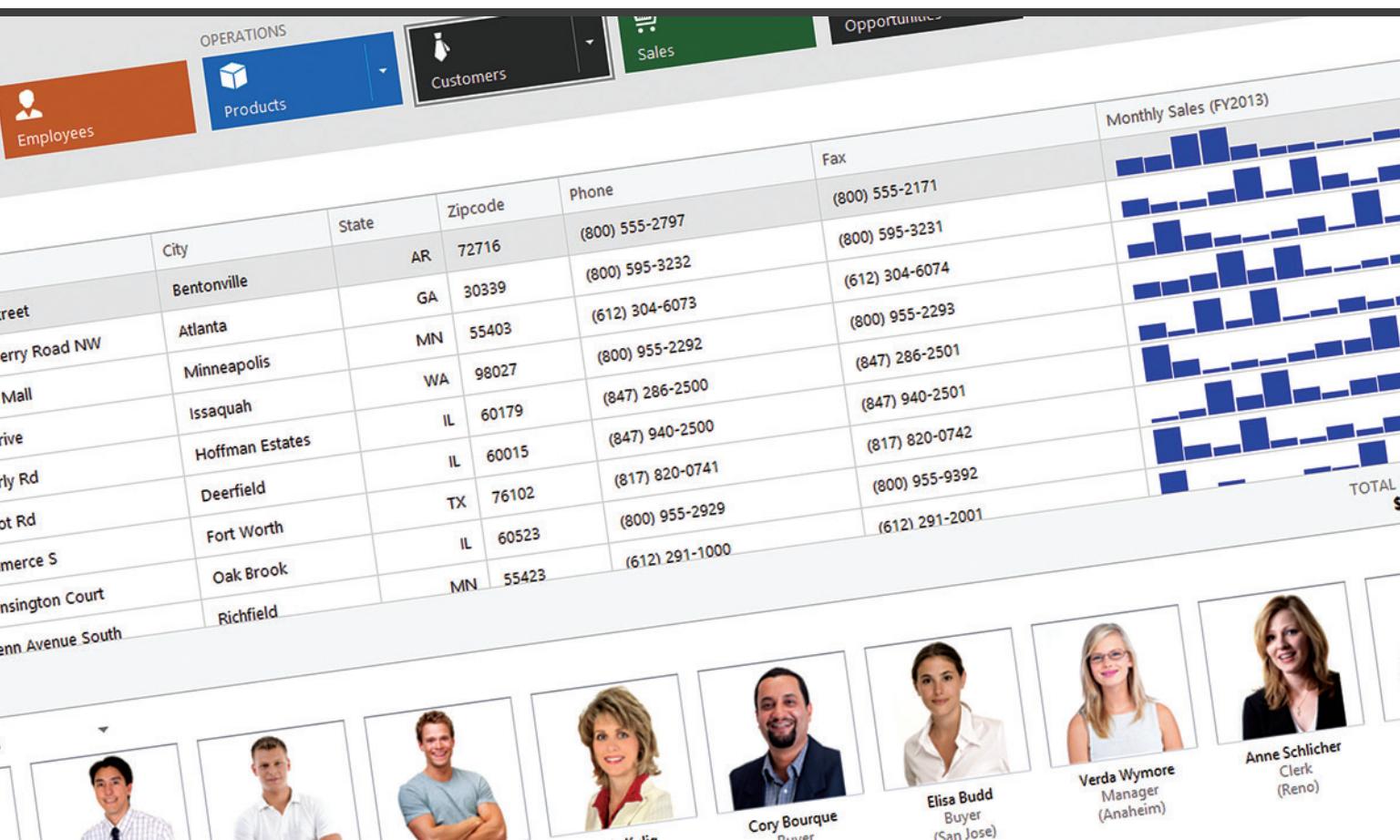
Finding wins allowed me to start appreciating the many small steps I took toward success in my mega-projects. Previously, I'd feel demotivated and inadequate when working on a long-term project; with completion weeks away, my little steps *now* seemed unimportant. I'd only intellectually appreciated that any project required a lot of work. By focusing on wins, I began to emotionally acknowledge all that I was pouring into it.

Within a year of beginning this inertia-blasting, goal-setting, win-finding journey, I turned from a frustrated drifter to a razor-focused success machine. In the words of one colleague, I became "a force of nature." ■

KRISHNAN RANGACHARI is a career coach for developers. He's held senior engineering roles at Silicon Valley startups and large companies like Microsoft (where he started working full time at age 18). Visit radicalshifts.com to download his free career success kit. Reach him directly at k@radicalshifts.com.

Touch-Optimized Hybrid Apps

DevExpress Universal provides a comprehensive collection of UX tools so you can create touch-first apps that are built for today and ready for tomorrow.



The screenshot displays a hybrid application interface. At the top, there's a navigation bar with tabs: Employees (selected), Products, Customers, Sales, and Opportunities. Below the navigation bar is a grid table with columns for City, State, Zipcode, Phone, Fax, and a large area for a monthly sales chart. The grid contains data for various cities like Bentonville, Atlanta, Minneapolis, etc. To the right of the grid is a detailed bar chart titled "Monthly Sales (FY2013)" showing sales volume over time. At the bottom of the screen, there's a horizontal gallery of seven user profile cards, each featuring a photo and the name and title of the employee.

	City	State	Zipcode	Phone	Fax
Street	Bentonville	AR	72716	(800) 555-2797	(800) 555-2171
erry Road NW	Atlanta	GA	30339	(800) 595-3232	(800) 595-3231
Mall	Minneapolis	MN	55403	(612) 304-6073	(612) 304-6074
ive	Issaquah	WA	98027	(800) 955-2292	(800) 955-2293
ry Rd	Hoffman Estates	IL	60179	(847) 286-2500	(847) 286-2501
ot Rd	Deerfield	IL	60015	(847) 940-2500	(847) 940-2501
merce S	Fort Worth	TX	76102	(817) 820-0741	(817) 820-0742
nsington Court	Oak Brook	IL	60523	(800) 955-2929	(800) 955-9392
enn Avenue South	Richfield	MN	55423	(612) 291-1000	(612) 291-2001



Your Next Great Hybrid App Starts Here

Build touch-first apps for any Windows device and leverage existing code investments.

devexpress.com/hybrid





Don't Gamble with UX—Use Wireframes

Paraphrasing immortal scientist Albert Einstein, I dare say, I don't know what the future has in store for software architecture, but if any changes are upcoming, one would certainly be embracing a top-down methodology in design of layers. Short for UX-driven design, UXDD is an approach that primarily promotes the use of wireframes and storyboards as the starting point of software architecture. In UXDD, the outcome of the elicitation process isn't a dry-and-flat collection of requirements, but instead an alive-and-kicking collection of sketches clearly indicating the way users love to work. And not surprisingly, the way users love to work is the plain software representation of real business processes. As I see things, technology is losing the central role in software architecture it had for decades. Today it takes more the architect's ability to combine available technologies and products to achieve a specific set of goals, taking into account existing investments in specific technologies, legacy platforms, skills and budgets.

Sounds like a foregone thought? Well, it probably is. This vision of software is nothing new, but it's also a vision of software that has never become mainstream.

Defining Wireframes

I've been using the word "wireframe" and also use it interchangeably with the word "sketch." Let me clarify and set in stone the difference, from a UXDD perspective, between wireframes and "mockups." Here are some definitions for terms in UXDD:

- A *sketch* is a freehand drawing primarily done to jot down and capture a flashing idea. Ideal surface to persist a sketch is paper, including paper napkins from cafeterias. A sketch may represent the top-level skeleton of an architecture, but, more likely, it captures the idea of an interaction between the user and the system.
- A *wireframe* is a more precise and accurate type of sketch as it adds information about the layout, navigation logic and details about the organization of the content. At the same time, a wireframe is agnostic of any UI details such as colors, graphics, shadows and fonts.
- When a wireframe is extended to incorporate graphical skins, then it becomes a *mockup*.

There's a key psychological difference between mockups and wireframes. When you show users mockups of an application, the attention is inevitably captured by graphical aspects such as background images, colors and style. Rarely, the users' attention is caught by organization of the information and how the same information is navigated and laid out. Wireframes are kind of scanty mockups

entirely devoid of graphical content. Wireframes can be drawn manually or through generic drawing tools such as PowerPoint, Visio or, perhaps, Balsamiq. More specialized tools, however, are appearing to link wireframes together, thus forming storyboards to give users a concrete idea of the final experience.

Why Wireframes Are Cost-Effective

Developers generally agree that using wireframes to form an idea of the presentation layer before even planning the back end is greatly beneficial. For many years, folks in this industry cultivated the idea that architecting software is much the same as architecting buildings. The Agile movement then emerged to combat the idea that we needed a big up-front and closed design before writing code. Not that Agile really means no design and no analysis—quite the reverse, I'd say—but Agile probably failed addressing the core issue of software development. What is the ultimate goal of software? Are we simply writing software just to store and handle data? Instead, aren't we writing software primarily to have users perform business tasks more efficiently and more quickly? In the latter scenario, of growing relevance today, the presentation layer is all that matters. Agility in this case means iterating and collaborating with users to define an acceptable set of wireframes and storyboards rather than information sufficient to create a good persistence model.

A *sketch* is a freehand drawing primarily done to jot down and capture a flashing idea.

Yet it's not uncommon for software managers to see any time spent on wireframes as a useless cost that brings no deliverables and no direct money. As such, it can be avoided to have the team focus on actual software artifacts. The analogy between software and civil engineering is a longtime analogy and it's useful to recall it now to say the following: When you want to build a garage (not to mention more sophisticated types of buildings), are you going to explain what you want or are you just expecting bricklayers to put bricks together and then complain when they deliver something you never wanted or asked for?

Creating and validating wireframes is an extra task to manage and an extra cost. However, it's one thing to tweak a wireframe file that jots down a visual screen; it's quite another to tweak a multi-layered

We didn't invent the Internet...

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



TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

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



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

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

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

connectivity

powered by 

To learn more please visit our website →

www.nsoftware.com

system made of thousands of lines of code. By adopting wireframes and UXDD, you definitely spend more time before any single line of code is written, but you save a lot more time once the first sprint is delivered. And you save even more time when the system is halfway and its effects and benefits can be better experienced and evaluated. With UXDD, you go much closer than ever to the dream of delivering the project right away. What you save in subsequent sprints and remakes is much more than you spend up front with wireframes.

When a wireframe is extended to incorporate graphical skins, then it becomes a *mockup*.

How to Produce Wireframes

At the foundation of UXDD and wireframes lies the idea that software should not be created to model the business domain. Rather, it should be written to *mirror* the business domain. This shift of perspective has relevant spin-offs in the patterns we use to devise the system. In particular, it gives the preliminary analysis a strongly task-oriented focus and blurs the relevance of modeling. This aspect scares most managers because it sounds like a weird thing to do that nobody did for decades. The expected outcome of interviews with customers has long been information to create an all-encompassing data model to read and update the state of the system. As complexity of domains and applications grows, this approach is less and less effective and manageable.

Classic individual and group interviews are still acceptable, but more desirable if aimed at understanding the relevant events to handle in the system and the tasks and circumstances that cause them. Event storming is an emerging methodology to capture wireframes

and to lay out the presentation layer quickly and effectively. You can find a brief introduction to the practice of event storming, written by the very own author and master, at bit.ly/1N25MJI. In a nutshell, this is a method to explore complex business domains in a way that is both quick and straight to the point. It has a strong task-oriented flavor and lets you learn about the domain and, especially, the tasks to implement. Better yet, it lets you listen to what users really need to do and the actual flow of actions, triggers, and events you need to mirror and reflect through software.

How to Validate Wireframes

This is a key point: Without an explicit signoff and acceptance from the customer on the wireframe creation, any effort is lost because it doesn't guarantee a final ideal UX. In UXDD, you must have approval for wireframes before you start writing any serious amount of code.

Having customers look at wireframes is only the first step, though. Effective validation of wireframes typically requires more effort. You must show users what it means to perform a task and the resulting sequence of steps and screens that completes any given task. Storyboard is the term used to refer to the concatenation of multiple wireframes to compose a business task. If the content of wireframes is accurate enough, playing a storyboard is really close to testing a software-made prototype, except that the cost of a storyboard is a fraction of the cost of a software prototype and can be fixed nearly instantaneously. In more advanced scenarios, it might even be reasonable for film users to use film while playing storyboards. Their body language may reveal a lot, as well as the average time that each operation takes. An accurate validation process might also include elements of cognitive analysis such as Test-Operate-Test-Exit (TOTE). As shown in **Figure 1**, TOTE is essentially an iterative trial-and-error session aimed at progressively improving the satisfaction of users and achievement of shared goals.

There's no magic behind TOTE—it's plain common sense, but as foregone as it sounds, it works effectively. The more you apply cognitive analysis and the more you spend up front, the more you save later. For those who do so, the balance is largely positive. Overall, it can't be worse than when you start coding based on assumed models and then figure out the wrong artifacts were delivered.

An Exemplary Story

The story that follows is real. After an endless series of meetings, both the team and the customer were sure that all aspects of the Web site to build had been properly discussed and analyzed. Every single aspect of the system was allegedly clarified and written in stone; no bad surprises were in order. The honeymoon lasted for just a couple of days until the developer working on the login page raised the first exception. Is login going to happen only through membership? Would there be no social authentication? The answer was a resounding "No, the customer didn't mention that." The login was delivered without Facebook and Twitter authentication. Guess what? Social logins were for the customer a strict requirement; it was so strict and obvious that it wasn't even mentioned. Social logins were extra work and knowing that beforehand would have likely oriented the implementation of the membership system differently. At the same time, it was clear to everybody within the team that if anybody showed

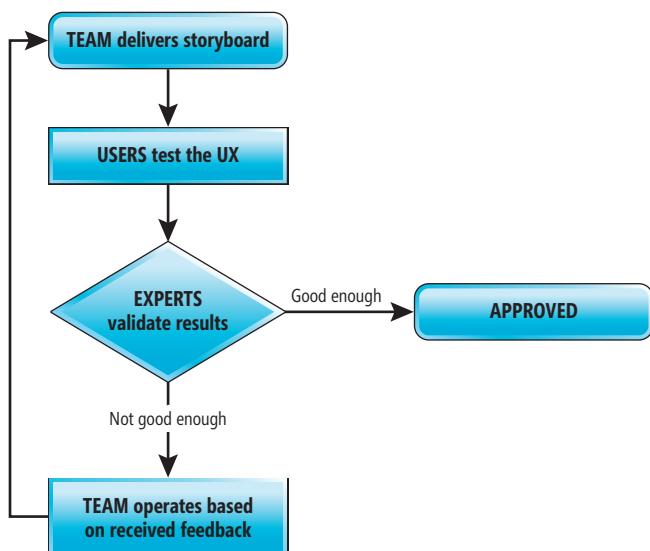


Figure 1 TOTE Flowchart from Cognitive Analysis Adapted to UX Analysis

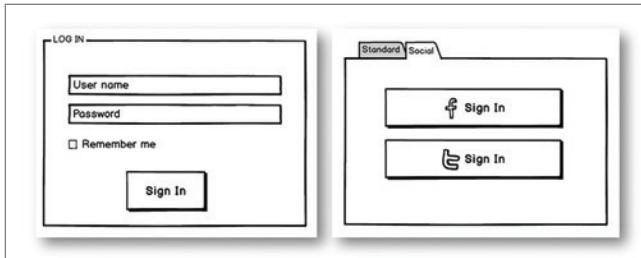


Figure 2 Wireframes for Social Login that Nobody Created

users a sketch of the login page the lack of the social button would have been noted and reported, as shown in Figure 2.

From Wireframes to Prototypes

The more you feel engaged by the idea of using cognitive analysis in the elicitation process, the more you might be looking for specific tools that help with creating wireframes and storyboards. Wireframes are relatively easy and quick to create, but the same can't be said for storyboards. It should be noted that the simple concatenation of a few wireframes may not be enough for certain users and certain business scenarios. Sometimes, some code must be written to proof concepts. Some terminology applies here that is useful to clarify. A proof-of-concept is typically a small exercise to verify an assumption or play with a new technology in a given context. I like to call a proof-of-concept a domain-specific "hello world." A prototype instead is a fake system that attempts to simulate the full system and is done to test the viability or even the usefulness of a system. This is what you need to have when storyboards prove not to be enough. Finally, another term that is often used interchangeably while having its own strict definition is "pilot." A pilot is a full production system only tested against a subset of the general intended audience or data set.

Wrapping Up

To be effective, wireframes should be the currency to exchange between stakeholders, just at a different abstraction level than the domain-drive design ubiquitous language. Both ubiquitous language and wireframes aim at simplifying the conversation and reducing the risk of misunderstanding, misrepresentation of data, and wrong assumptions because of lack of information and guidance.

Ubiquitous language and wireframes are not methods that can magically turn ideas into running code. Far less ambitiously, the point of both is to find a sequence of concrete steps that can reduce the cost of writing software close to expected budget and deadlines.

Next month I'll take this further and discuss the (deep) architectural implications of using wireframes. ■

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

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

msdnmagazine.com



dtSearch®

Instantly Search Terabytes of Text

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

Highlights hits in all data types;
25+ search options

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

Visit [dtSearch.com](http://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



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



EF7 Migrations: Not New but Definitely Improved

Code First Migrations has undergone some dramatic changes with Entity Framework 7. While the basic concepts and migration steps remain the same, the workflow has become much cleaner and more straight-forward, and it now works nicely with source control.

EF7 brings two flavors of migrations: the familiar variety you use with NuGet and most .NET projects, and the kind you run as part of the DNX Runtime with ASP.NET 5 apps.

I took a look at these in one of my Pluralsight courses (bit.ly/1MThS4J). Though a few things have changed between the beta 4 version I used in that course and the Release Candidate 1 (RC1) I'm using today, that course remains a great resource for seeing the migrations in action and learning about some of the differences.

But with the release of RC1, which is considered feature complete for the upcoming RTM, it was time to take another close look at EF7 migrations. I'll begin with the familiar Windows PowerShell commands in the Package Manager Console of Visual Studio. I'll then discuss these commands as you'd use them with DNX and ASP.NET 5. This article presumes you have some existing knowledge of the database initialization and migrations features in EF6 or earlier versions.

No More Magic Database Initialization— Use Migrations

Supporting magic is expensive, limiting and creates a lot of baggage. And in the case of DbInitializers, it created a lot of confusion. So EF DbInitializers, along with the default behavior of CreateDatabaseIfNotExists, are now gone. Instead, there's logic that lets you explicitly create and delete databases (EnsureDeleted, EnsureCreated), which I'll be happy to use in integration tests.

EF7 is designed to rely on migrations. They should now be your default workflow.

The “enable-migrations” command existed because migrations were tacked onto EF and using them required building out a bit of infrastructure. That command didn't install any new APIs; it just

added a project folder and a new DbMigrationConfiguration class to your project. With EF7, the relevant logic is internal and you don't have to explicitly tell EF you want to use migrations.

Supporting magic is expensive, limiting and creates a lot of baggage, so the DbInitializers along with the default behavior of CreateDatabaseIfNotExists are now gone.

Remember—EF7 is composable. Not everyone will want or need migrations, so if you do, you need to opt in to the migrations commands by installing the package that contains them:

```
install-package entityframework.commands -pre
```

Cmdlet	Description
Add-Migration	Adds a new migration.
Remove-Migration	Removes the last migration.
Scaffold-DbContext	Scaffolds a DbContext and entity type classes for a specified database.
Script-Migration	Generates a SQL script from migrations.
Update-Database	Updates the database to a specified migration.
Use-DbContext	Sets the default DbContext to use.

Figure 1 The Migrations Commands as Listed Via the NuGet Package Manager Get-Command

```
SYNTAX
    Update-Database [[-Migration] <String>] [-Context <String>] [-Project <String>] [-StartupProject <String>] [<CommonParameters>]

PARAMETERS
    -Migration <String>
        Specifies the target migration. If '0', all migrations will be reverted. If omitted, all pending migrations will be applied.

    -Context <String>
        Specifies the DbContext to use. If omitted, the default DbContext is used.

    -Project <String>
        Specifies the project to use. If omitted, the default project is used.

    -StartupProject <String>
        Specifies the startup project to use. If omitted, the solution's startup project is used.
```

Figure 2 Update-Database Syntax and Parameters

```

PARAMETERS
-From <String>
    Specifies the starting migration. If omitted, '0' (the initial database) is used.

-To <String>
    Specifies the ending migration. If omitted, the last migration is used.

-Idempotent [<SwitchParameter>]
    Generates an idempotent script that can be used on a database at any migration.

-Context <String>
    Specifies the DbContext to use. If omitted, the default DbContext is used.

-Project <String>
    Specifies the project to use. If omitted, the default project is used.

-StartupProject <String>
    Specifies the startup project to use. If omitted, the solution's startup project is used.

```

Figure 3 Script-Migration Parameters

The EF7 Migrations Commands

Using the Windows PowerShell Get-Help cmdlet displays the current (as of RC1) migration commands, as shown in **Figure 1**. This list should be the same for the RTM.

Note also that Update-Database no longer has the “-script” parameter. It will do only what its name suggests: update the database. As always, using this command on production databases isn’t recommended. **Figure 2** displays details about the Update-Database command.

Automatic migrations has been removed from EF7. Thanks to that, managing migrations is simpler and source control is now supported.

To create scripts, you can use the new Script-Migration command. And there’s no longer a secret recipe for creating idempotent scripts (introduced in EF6). Idempotent is a parameter, as you can see in **Figure 3**.

Big Changes for Migration History and Model Snapshots

Automatic migrations has been removed from EF7. Thanks to that, managing migrations is simpler and source control is now supported.

Figure 4 The Complete newStringInSamurai Migration Class

```

public partial class newStringInSamurai : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.AddColumn<string>(
            name: "MyProperty",
            table: "Samurai",
            nullable: true);
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropColumn(name: "MyProperty", table: "Samurai");
    }
}

```

In order for migrations to determine what needs to happen, a history of previous migrations has to exist. Each time you run the add-migration command, the API will look at that historical information. Previously, migrations stored an encoded version of a snapshot of the model for each migration in the database, in a table (most recently) named _MigrationHistory. This is what EF has used to figure out what a

migration needed to accomplish. Each time add-migration was run, it meant a trip to the database to read this history, and that was the source of a lot of problems.

With EF7, when a migration is created (via add-migration), in addition to the familiar migration file inside the Migrations folder, EF7 also creates a file to store the current model snapshot. The model is described using the Fluent API syntax used for configurations. If you make a change to your model and then add a migration, the snapshot gets revised with the current description of the full model. **Figure 4** shows a migration file that introduces a new string called “MyProperty.”

And here’s the portion of the updated ModelSnapshot class, where you can see that “MyProperty” is now part of the description of the Samurai class:

```

modelBuilder.Entity("EF7Samurai.Domain.Samurai", b =>
{
    b.Property<int>("Id")
        .ValueGeneratedOnAdd();
    b.Property<string>("MyProperty");
    b.Property<string>("Name");
    b.HasKey("Id");
});

```

So, individual migrations describe what has changed, as they always have. But now, in the project, you always have a description of the full model. This means that when you add a new migration, EF doesn’t have to go to the database to find out what the previous model looked like. What it needs is right at hand. And even more important, this works much better with source control because it’s right there in your code and can be diffed and merged by source control like any other file. It’s not tucked away in a database that may not even be accessible to your teammates. Before EF7, there was no good way to manage migrations in source control. You’ll find guidance about migrations for versions earlier than EF7 in “Code First Migrations in Team Environments” (bit.ly/10V03qr). This article begins with the advice: “Grab a coffee, you need to read this whole article.”

With EF7, the migration history table in the database (**Figure 5**) now stores only the migration’s id and the version of EF that created it. Again, this is a big change from earlier versions, which stored the snapshot itself into this table. Update-Database will check this table to see which migrations have been applied and applies any that are missing, even those that aren’t consecutive. As long as this operation needs to touch the database, I see no problem with migrations doing a quick pre-check against the database in this scenario.

The screenshot shows the SQL Server Object Explorer on the left with a tree view of databases, including '(localdb)\MSSQLLocalDB'. On the right, a table named 'dbo_EFMigrationsHistory' is displayed with two rows of data. The columns are 'MigrationId' and 'ProductVersion'. The first row is '20151018211350_initial' and '7.0.0-beta8-15964'. The second row is '20151019171819_newStringInSamurai' and '7.0.0-beta8-15964'. A note at the bottom of the table says 'NULL'.

Figure 5 The dbo_EFMigrationsHistory Displaying Migrations Added Table

You no longer have to deal with that nonsense about not being able to add a migration if the previous one hasn't been applied to the database yet. That was a side effect of automatic migrations and caused many of us to stumble into a tangled web of confusion. Now you can build up migrations in your solution as needed and then update the database when you're ready.

You no longer have to deal with that nonsense about not being able to add a migration if the previous one hasn't been applied to the database yet.

Because of the model snapshot that EF7 stores in the project, you shouldn't just delete a migration file from your project because the snapshot would then be incorrect. So, along with the change to the migrations workflow comes a new command: remove-migration. Remove-migration is designed to undo a migration that you most likely just added and then changed your mind about before it was applied to the database. The command does two things in your project: It removes the latest migration file and it then updates the snapshot file in the migrations folder. Interestingly, this doesn't rebuild the snapshot based on the model. Instead, it modifies the snapshot based

Figure 6 Dealing with an Unwanted Migration

State of Database	Commands	Tasks Performed by Command
Update-Database not run after Migration2	Remove-Migration	<ol style="list-style-type: none"> Verifies Migration2 is not in history table Deletes Migration2.cs Fixes Snapshot.cs to reflect previous migration
Update-Database was run after Migration2	Update-Database Migration1	<ol style="list-style-type: none"> Runs SQL for "Drop" method of Migration2 Removes Migration2 row from history table
	Remove-Migration	<ol style="list-style-type: none"> Deletes Migration2.cs Fixes Snapshot.cs to reflect previous migration

on the model snapshot tucked away in the Designer.cs file that's created along with each snapshot. I tested this out and saw that if I didn't change the model (that is, I didn't remove the unwanted property from the Samurai class), the snapshot was still fixed up according to the migrations. But don't forget to fix your model, too. Otherwise the model and the snapshot will be out of sync and, most likely, your database schema will also be out of sync with the model.

As a side note, Brice Lambson, the EF team member responsible for migrations, shared with me that if you do delete a migration file manually, the very next call to remove-migration will see that and fix up the snapshot without deleting another migration file.

Like add-migration, remove-migration doesn't affect the database. However, it does check in the database's migration history file to see if the migration has already been applied. If it has, then you can't use remove-migration, not yet at least.

Sounds confusing, I know. But I still like remove-migration because the alternative is to just keep moving forward and adding a migration whenever you change your mind. That's potentially confusing to other team members or to the future you. Plus, it's a small price to pay for the benefit of allowing migrations to participate in source control.

To help clarify, **Figure 6** has guidance for unwinding an unwanted migration. The chart presumes that "Migration1" has been applied to the database and "Migration2" was created, but never applied to the database.

If you need to undo multiple migrations, start by calling update-database followed by the name of the last good migration. Then you could call remove-migration to roll back migrations and the snapshot, one at a time.

With all of this focus on the migrations, don't forget that those migrations are a reflection of changes to the model. So here is one last reminder that you must manually undo the changes to the model and then remove any migrations that reflect those changes.

Reverse Engineer with Scaffold-DbContext

Previous versions of EF provided the ability to reverse engineer a database into Code First models. Microsoft made this capability available through the Entity Framework Power Tools and, starting with EF6, through the EF Designer. There are also third-party tools such as the popular (free) extension, EntityFramework Reverse POCO Generator (reversepoco.com). Because EF7 doesn't have a designer, the scaffold command was created to do this job.

Here are the parameters for Scaffold-DbContext:

```
Scaffold-DbContext [-Connection] <String> [-Provider] <String>
[-OutputDirectory <String>] [-ContextClassName <String>]
[-Tables <String>] [-DataAnnotations] [-Project <String>]
[<CommonParameters>]
```

I'm a big fan of fluent API configurations, so I'm happy that these are once again the default and you need to opt in to using data annotation instead with the DataAnnotations flag. The following

1&1 CLOUD SERVER

TEST THE BEST!



Powered by

Cloud
Technology

Easy to use – ready to go.

The 1&1 Cloud Server provides superior CPU, RAM, and SSD performance. Give your cloud projects the perfect combination of flexibility and efficient features.

- ✓ Load balancing
- ✓ SSD storage
- ✓ Billing by the minute
- ✓ Intel® Xeon® Processor E5-2660 v2 and E5-2683 v3



1 month free!
Then from \$4.99/month.*



1 (877) 461-2631



1and1.com

* 1&1 Cloud Server is available free for one month, after which regular pricing starts from \$4.99/month. No setup fee is required. Visit 1and1.com for full offer details, terms and conditions. Intel, the Intel Logo, Intel Inside, the Intel Inside logo, Intel Experience What's Inside are trademarks of Intel Corporation in the U.S. and/or other countries. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are property of their respective owners. ©2015 1&1 Internet. All rights reserved.

is a command to reverse engineer from my existing database using that parameter (I could also select which tables to scaffold, but I won't bother with that here):

```
Scaffold-DbContext
  -Connection "Server = (localdb)\mssqllocaldb;
  Database=EF7Samurai;" 
  -Provider entityframework.sqlserver
  -OutputDirectory "testRE"
```

As **Figure 7** shows, this resulted in a new folder and several files to be added to my project. I have a lot more examining of those classes to do because this is an important feature, but I'll leave that for the future.



Figure 7 Result of Scaffold-DbContext

The DNX Versions of Migrations Commands

The commands you run in the Package Manager Console are Windows PowerShell commands, and Windows PowerShell is, of course, part of Windows. One of the most important features of ASP.NET 5—and EF7—is they are no longer dependent on Windows. DNX is a lightweight .NET execution environment that can be run on OS X and Linux, as well as Windows. You can read more about it at bit.ly/1Gu34wX. The entityframework.commands package contains migrations commands for the DNX environment, as well. Here's a quick look at those commands.

You can execute the commands directly in the Package Manager Console if you happen to be using Visual Studio 2015, which is what I'll show here. Otherwise, you execute them at the command line of your relevant OS. Check out Nate McMaster's Channel 9 video where he codes up EF7 and migrations on a Mac. (bit.ly/10SUCdo)

First, you need to be sure you're in the correct folder—that's the folder for the project where your model resides. The Default project dropdown isn't related to your file path. Use the dir command to see where you are and then just cd to the correct folder. Tab completion is your friend here. In **Figure 8** you can see that although the default project is src\EF7WebAPI, in order to execute dnx commands in that directory, I have to explicitly change the directory to the path for this project that contains my EF7 model.

Once there, I can execute commands. Every command, by the way, needs to be prefaced with dnx, and you can use dnx ef to list all of the available commands. Note that ef is a shortcut I've set up in project.json. Check out my Pluralsight video, "Looking Ahead to Entity Framework 7" (bit.ly/PS_EF7Alpha), to see more about the setup.

```
PM> cd ..\src\
Directory: D:\GitHubRepos_Random\EF7WhoAreYou\ASPNET5WebAPI

Mode          LastWriteTime    Length Name
----          -----        ---- 
d----

```

Figure 8 Getting to the Correct Directory Before Running DNX Commands

The core commands are database, dbcontext and migrations; migrations has the following sub-commands:

```
add      Add a new migration
list     List the migrations
remove   Remove the last migration
script   Generate a SQL script from migrations
```

Each command has parameters you can query by adding --help, like this:

```
dnx ef migrations add --help
```

For example, if the migration name is initial, then the command is:

```
dnx migrations add initial
```

Remember that these are the same commands you saw earlier in this article, so you'll find parameters that allow you to specify the project, the DbContext and other details. The script command has the idempotent parameter as described earlier. But one thing that's different about the dnx script command is that, by default, the script is simply listed in the command window. You need to explicitly specify an output <file> parameter to push the script to a file.

The following command will apply the migrations to the database:

```
dnx ef database update
```

The dbcontext command has two subcommands: dbcontext list will list all of the DbContexts that can be discovered in the project. This is here because you can't get tab-expansion to provide the options easily like you can when using the commands in Windows PowerShell; scaffold is the dnx version of the DbContext-Scaffold command you learned about earlier.

Overall a Much Better Migrations Experience with EF7

With the removal of automatic migrations, migrations have been simplified in EF7. Although the basic concepts are the same, the workflow is a lot cleaner. Removing the barriers to using migrations in a team environment was critical. I like the way this was fixed by bringing the model snapshot into the source code. And allowing reverse engineering with the scaffold command is a great way to provide this important capability without relying on a designer.

The July 23 Design Meeting Notes from the EF team (bit.ly/18S13ro) include a nice table with all of the Windows PowerShell and dnx commands after the naming was tightened up. I expect there will be an even cleaner version in the documentation eventually, but I still find this one useful.

I wrote this article using the RC1 version of EF7, but this version is considered to be feature complete, so it's likely this information will align with the release of EF7 that's coming along with ASP.NET 5 in early 2016. ■

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" (2009), 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 julielerman.me/PS-Videos.

THANKS to the following Microsoft technical expert for reviewing this article: Brice Lambson

Data entry errors. Consolidation headaches. Quality data shouldn't be a pain.

Try a little Vitamin Q.



We supply Vitamin Q – “quality” – in data quality management solutions that profile, clean, enrich, and match your customer data – and keep it healthy over time. Add Vitamin Q to your data integration, business intelligence, Big Data, and CRM initiatives to ensure accurate data for greater insight and business value.

Get a **FREE 30-day trial!**
www.MelissaData.com/vitaminQ



Solutions for
240+ Countries



10,000+ Customers
Worldwide



30+ Years
Strong



Data Quality &
Mailing Solutions



Cloud • On-Premise
• Services



Germany
www.MelissaData.de

United Kingdom
www.MelissaData.co.uk

India
www.MelissaData.in

Australia
www.MelissaData.com.au

MELISSA DATA®

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

Making Big Data Batch Analytics Easier Using U-SQL

Michael Rys

The new Microsoft Azure Data Lake services for analytics in the cloud (bit.ly/1VcCkaH) includes a hyper-scale repository; a new analytics service built on YARN (bit.ly/1iS8xvP) that lets data developers and data scientists analyze all data; and HDInsight (bit.ly/1KFywqg), a fully managed Hadoop, Spark, Storm and HBase service. Azure Data Lake Analytics also includes U-SQL, a language that unifies the benefits of SQL with the expressive power of your own code. The U-SQL scalable distributed query capability enables you to efficiently analyze data in the store and across relational stores such as Azure SQL Database. In this article, I'll outline the motivation for U-SQL, some of the inspiration and design philosophy behind the language and give a few examples of the major aspects of the language.

Why U-SQL?

If you analyze the characteristics of Big Data analytics, several requirements arise naturally for an easy-to-use, yet powerful language:

- Process any type of data. From analyzing BotNet attack patterns from security logs to extracting features from images and videos for machine learning, the language needs to enable you to work on any data.

This article discusses:

- The advantages of Big Data language U-SQL
- How easy it is to use C# to add user code to U-SQL
- How to use Visual Studio to speed up U-SQL development
- How you can use U-SQL capabilities to store and analyze structured and unstructured data

Technologies discussed:

Azure Data Lake Tools for Visual Studio, U-SQL, C#, Azure Data Lake

- Use custom code easily to express complex, often proprietary business algorithms. The example scenarios in the first bullet point might all require custom processing that's often not easily expressed in standard query languages, ranging from user-defined functions, to custom input and output formats.
- Scale efficiently to any size of data without focusing on scale-out topologies, plumbing code or limitations of a specific distributed infrastructure.

How Do Existing Big Data Languages Stack Up to These Requirements?

SQL-based languages such as Hive (hive.apache.org) provide a declarative approach that natively does scaling, parallel execution and optimizations. This makes them easy to use, familiar to a wide range of developers, and powerful for many standard types of analytics and warehousing. However, their extensibility model and support for non-structured data and files are often bolted on and more difficult to use. For example, even if you just want to quickly explore data in a file or remote data source, you need to create catalog objects to schematize file data or remote sources before you can query them, which reduces agility. And although SQL-based languages often have several extensibility points for custom formatters, user-defined functions, and aggregators, they're rather complex to build, integrate and maintain, with varying degrees of consistency in the programming models.

Programming language-based approaches to process Big Data, for their part, provide an easy way to add your custom code. However, a programmer often has to explicitly code for scale and performance, often down to managing the execution topology and workflow such as the synchronization between the different execution stages or the scale-out architecture. This code can be difficult to write correctly and even harder to optimize for

Figure 1 Data Stream Preview in Visual Studio

performance. Some frameworks support declarative components such as language-integrated queries or support for embedded SQL. However, this SQL may be integrated as literal strings lacking tool support. Therefore, the extensibility integration might be limited or—due to the procedural code that doesn't guard against side effects—hard to optimize and difficult to reuse.

Satisfying the Requirements with U-SQL

Taking the issues of both SQL-based and procedural languages into account, U-SQL was designed from the ground up as an evolution of the declarative SQL language with native extensibility through user code written in C#. U-SQL unifies the following:

- The declarative and imperative coding paradigms and experiences
- The experience around extending your language capabilities with custom user code
- Processing over all data, whether unstructured or structured
- Data processing in the Azure Data Lake and other Azure data sources using federated queries

U-SQL is built on the learnings from the Microsoft internal experience with a declarative and extensible scripting language (bit.ly/10GUNIY) called Structured Computations Optimized for Parallel Execution, or SCOPE, and existing languages such as T-SQL, ANSI SQL and Hive. For example, we base our SQL and programming language integration and the execution and optimization framework for U-SQL on SCOPE, which currently runs hundreds of thousands of jobs each day internally. We also align the metadata system (databases, tables and so on), the SQL syntax and language semantics with T-SQL and ANSI SQL, the query languages with which most of our SQL Server customers are familiar. And we use C# data types and the C# expression language so you can seamlessly write C# predicates and expressions inside SELECT statements and use C# to add your custom logic. Finally, we looked to Hive and other

Big Data languages to identify patterns and data processing requirements and integrate them into our framework.

In short, basing U-SQL language on these existing languages and experiences should represent a true evolution in the Big Data query language space and make it easy for you to get started, yet powerful enough for the most difficult problems.

Show Me U-SQL

Let's assume that I've downloaded my Twitter history of all my tweets, retweets and mentions as a CSV file and placed it into my Azure Data Lake store. I can use Azure Data Lake Tools for Visual Studio to preview the file and understand the structure. I can do this using the Server Explorer to find my Data Lake Storage accounts and open the explorer on my default storage account. **Figure 1** shows a data stream

(in this case my CSV file). In U-SQL I use a late binding of format to my data streams, so this preview just shows columns based on a delimiter, but doesn't specify type.

In this case, I know the schema of the data I want to process and for starters I want to just count the number of tweets for each of the authors in the tweet "network." The U-SQL script in **Figure 2** shows the three major steps of processing data with U-SQL:

1. Extract data from your sources into rowsets. Note that you just schematize it in your query with the EXTRACT statement. The datatypes are based on C# datatypes and the script uses the built-in Extractors library to read and schematize the CSV file.
2. Transform the rowset using U-SQL or custom-defined operators in a series of rowset manipulations, each often building on one or more past rowsets. In the example in **Figure 2**, it's a familiar SQL expression that does a GROUP BY aggregation on the rowset generated by the EXTRACT expression.
3. Output the resulting rowsets either into files or into U-SQL tables to store it for further processing.

Figure 2 U-SQL Script for Extracting Tweets from CSV

```

1 @t = EXTRACT date string
2           , time string
3           , author string
4           , tweet string
5   FROM "/input/MyTwitterHistory.csv"
6   USING Extractors.Csv();
7
8 @res = SELECT author
9           , COUNT(*) AS tweetcount
10      FROM @t
11     GROUP BY author;
12
13 OUTPUT @res TO "/output/MyTwitterAnalysis.csv"
14 ORDER BY tweetcount DESC
15 USING Outputters.Csv();

```

Note that SQL keywords in U-SQL have to be uppercase to provide syntactic differentiation from syntactic C# expressions with the same keywords but different meaning. A common error in early scripts is to use “as” instead of “AS” for assigning an alias to a column in my SELECT statement. The correct form is the uppercase AS, which is the SQL expression syntax. Of course, the error will be caught at compile time.

Also notice that each of the expressions is assigned to a variable (@*t* and @*res*). This lets U-SQL incrementally transform and combine data step-by-step expressed as an expression flow using functional lambda composition (similar to what you find in the Pig [pig.apache.org] language). The execution framework, then, composes the expressions together into a single expression. That single expression can then be globally optimized and scaled out in a way that isn’t possible if expressions are being executed line by line. **Figure 3** shows the execution graph generated for one of the later queries in this article. The graph represents the globally optimized execution stages arrived at by the compiler and optimizer.

C# Integration in U-SQL

Going back to my example, I now want to add additional information about the people mentioned in the tweets and extend my aggregation to return how often people in my tweet network are authoring tweets and how often they’re being mentioned. For this I’m taking a look at C# integration in U-SQL.

The core reliance of U-SQL on C# for its type system and scalar expression language provides the query writer access to the wealth of the C# and CLR libraries of classes, methods, functions, operators and types. All C# operators except for the assignment operators (=, += and so on) are valid in U-SQL scalar expressions. In particular, all comparison operators such as ==, !=, <, >, the ternary comparison cond ? true-expression : false-expression, the null coalesce operator ?? are supported. Even lambda expressions using => can be used inside U-SQL expressions.

This very tight integration and seamless programming experience is syntactically also enabled by U-SQL being integrated with the C# Roslyn compiler (bit.ly/1BsPced). In fact, the U-SQL integration is probably the most complex application of the C# Roslyn compiler platform.

There are several ways how C# code can be used to extend U-SQL expressions:

- Provide inline C# expressions in your U-SQL script: This often makes sense if a small set of C# methods needs to be applied to process one of the scalar values—a string type method or a math function.
- Write user-defined functions in a C# assembly and reference them in the U-SQL script: This is preferred for more complex functions, if the logic of the function requires the full power of C# beyond its expression language, such as procedural logic or recursion.

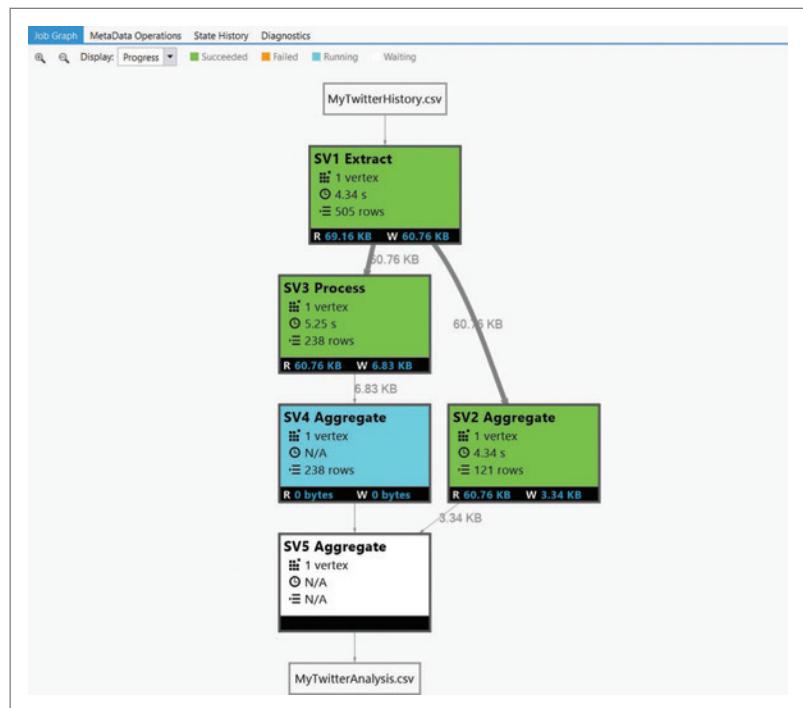
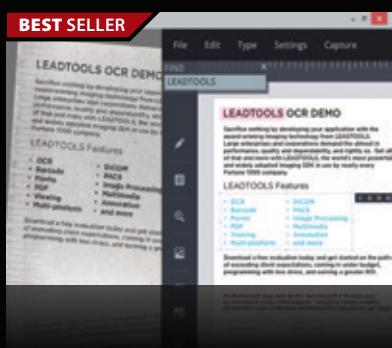


Figure 3 Job Execution Graph in Visual Studio

- Write user-defined aggregators in a C# assembly and reference them in the U-SQL script: By providing user-defined aggregators, custom aggregation logic can be plugged into U-SQL’s processing of aggregation with a GROUP BY clause.
- Write user-defined operators in a C# assembly and reference them in the U-SQL script: User-defined Operators (UDO) are U-SQL custom-coded rowset operators. They’re written in C# and provide the ability to generate, process and consume rowsets.

For the user-defined functions, aggregators and operators, the C# assembly will have to be loaded into the U-SQL metadata catalog with CREATE ASSEMBLY (U-SQL) and then referenced in the script with REFERENCE ASSEMBLY. The Azure Data Lake Tools for Visual Studio makes the registration process easy and even provides a so-called codebehind experience where you just write the code into a special C# file attached to a given script and on submission the tool takes care of all the plumbing.

Figure 4 shows an expanded query for the aggregation I just mentioned. I use a SELECT statement with an inline C# LINQ expression to extract the “mentions” from each tweet into an ARRAY in lines 8 to 10. @*m* (line 8) now contains a rowset of ARRAYS. I use the EXPLODE function in line 14 to turn each array into a rowset containing one row per array item. I do this EXPLODE as part of a CROSS APPLY, which means that it will be applied to each row of the rowset @*m*. The resulting new rowset (line 12) contains one “mention” per row. Note that I reuse the same name @*m*. I need to drop the leading @ sign to align it with my existing author values. This is done with another C# expression where I take the Substring starting at position 1 in line 12. Finally, I union the authors with the mentions in lines 16 to 21, and extend my COUNT aggregation to group by a case-insensitive Twitter handle and the category (lines


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


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


DevExpress DXperience 15.2 | from \$1,439.99


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


Help & Manual Professional | from \$586.04


Help and documentation for .NET and mobile applications.

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


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


Every Aspose .NET component in one package.

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

Figure 4 Tweet Manipulation Using C# Methods

```

1 @t = EXTRACT date string
2           , time string
3           , author string
4           , tweet string
5   FROM "/input/MyTwitterHistory.csv"
6   USING Extractors.Csv();
7
8 @m = SELECT new SQL.ARRAY<string>(
9           tweet.Split(' ').Where(x => x.StartsWith("@")))
10      AS refs
10  FROM @t;
11
12 @m = SELECT r.Substring(1) AS r
13           , "referenced" AS category
14   FROM @m CROSS APPLY EXPLODE(refs) AS t(r);
15
16 @t = SELECT author, "authored" AS category
17   FROM @t
18 UNION ALL
19   SELECT *
20   FROM @m
21 WHERE r != null && r != "";
22
23 @res = SELECT author.ToLowerInvariant() AS author
24           , category
25           , COUNT(*) AS tweetcount
26   FROM @t
27 GROUP BY author.ToLowerInvariant(), category;
28
29 OUTPUT @res TO "/output/MyTwitterAnalysis.csv"
30 ORDER BY tweetcount DESC
31 USING Outputters.Csv();

```

23 to 27) before outputting the result ordered by descending tweet count in lines 29 to 31.

Let's look at some of these parts in more detail to see the power of closely integrating C# into U-SQL.

The bolded parts of the script in **Figure 4** show some of the places where U-SQL expects and accepts C# expressions. As you can see, you can use the C# expressions in the EXTRACT and OUPUT USING clause, in the SELECT clause and WHERE clause, as well as in the GROUP BY clause, ORDER BY clause and EXPLODE function, although in this example I just refer to a column name in the two latter cases.

An important aspect of the integration is that the C# expressions have full access to the scalar values inside the query expression in a seamless way due to the fact that they're typed with C# types. For example, the columns *tweet* in line 9, *r* in lines 12, 21, and *author* in lines 23 and 27 are all seamlessly integrated into the C# expression without the need for extra wrapper syntax.

The EXTRACT and OUPUT USING clauses in lines 6 and 31 take C# expressions resulting in a user-defined operator instance. The two built-in expressions are calls to factory methods that return an extractor and outputter instance, respectively.

Let's look at the C# expression from lines 8 and 9 in a bit more detail:

```
new SQL.ARRAY<string>(tweet.Split(' ').Where(x => x.StartsWith("@)))
```

This is a great example of the use of C#: The U-SQL built-in type SQL.ARRAY<T> is actually a C# object type that provides the expected SQL/Hive capabilities without the side-effecting update functions of the existing C# Array type. You just use the new C# operator to create a new instance. The instance gets created by simply applying one of the many string operations on the column *tweet* that has been defined with the string type to break it into words. There's no more wondering where you get a certain string-type functionality as in normal SQL dialects: You have the full power of the CLR at your fingertips.

It gets even better. The *Split* method returns an *IEnumerable<string>*. So any further processing, such as filtering to get the "mentions" from the tweet words can be done with a LINQ expression and using a lambda expression as the predicate. Now try that with your standard SQL language!

Let's also look at the WHERE clause in line 21. Again, I can simply provide the C# expression referring to the columns in the rowset. The expression in this context has to result in a value of type bool. Now C# has two-valued logic and not three-valued logic the way SQL does it. Thus, the comparison *r != null* will return true if *r* is not null or false if it is null. By using the C# logical operator *&&*, I get the guarantee of the C# execution order being preserved and, more important, the short-cutting that will not execute the right comparison if the first will evaluate to false. U-SQL also supports the SQL-based AND and OR conjunctions that do not provide short-cutting, but let the predicates be reordered for better performance. All of this gives the developer the choice between more efficient execution and semantic safeguards.

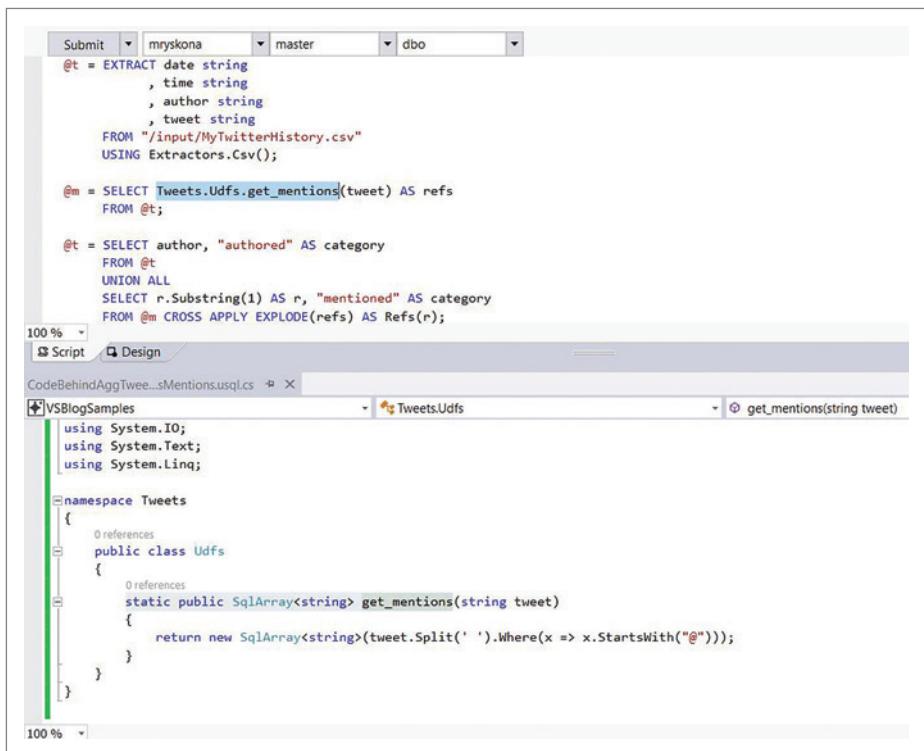


Figure 5 Codebehind in Visual Studio

WPF lives!



→ XCEED Business Suite for WPF

The essential set of WPF controls for all your line-of-business solutions. Includes the industry-leading Xceed DataGrid for WPF.
A total of 85 tools!

U-SQL, Visual Studio Codebehind Capabilities and Assemblies

As a next step I can use the Azure Data Lake Tools for Visual Studio to refactor the C# code into C# functions using the tool's codebehind functionality, as shown in **Figure 5**. When I then submit the script, it automatically deploys the code in the associated .cs file to the service on submission. In order to be able to refer to the methods and types and functions in U-SQL, the classes have to be defined as public and the objects need to be defined as static public.

The tool follows these three steps:

1. The .cs file is compiled into an assembly file.
2. The user's U-SQL Script gets augmented with a header that adds a CREATE ASSEMBLY statement that creates the assembly files binary content in your U-SQL metadata catalog.
3. It adds a cleanup at the end of the script that removes the registered assembly with a DROP ASSEMBLY statement.

I can also deploy and register the code as an assembly in my U-SQL metadata catalog myself explicitly. This lets me and other people use the code in future scripts. It's also the preferred way to manage your user-defined functions, aggregators and operators, if you have more complex code that you want to maintain separately, where you may want to include existing code that may have been written in other contexts (like your XML or JSON libraries) or even call out to external executables.

Similar to relational databases such as SQL Server, U-SQL provides a metadata catalog and supports the standard database objects such as databases, schemas, tables and so on. One of the objects is an assembly metadata object. By using the CREATE ASSEMBLY statement, you can register an assembly in the database. Assemblies are objects scoped to a database; the assembly DLL file gets placed into the assembly folder inside the relevant database folder inside the catalog folder in your primary Azure Data Lake storage account.

In addition to storing your assembly, you can specify additional files that will be stored together with your assembly and will be

Figure 6 Assembly References in U-SQL

```
1 REFERENCE ASSEMBLY TweetAnalysis;
2
3 @t = EXTRACT date string
4     , time string
5     , author string
6     , tweet string
7     FROM "/input/MyTwitterHistory.csv"
8     USING Extractors.Csv();
9
10 @m = SELECT Tweets.Udfs.get_mentions(tweet) AS refs
11     FROM @t;
12
13 @t = SELECT author, "authored" AS category
14     FROM @t
15     UNION ALL
16     SELECT Tweets.Udfs.cleanup_mentions(r) AS r, "mentioned" AS category
17     FROM @m CROSS APPLY EXPLODE(refs) AS Refs(r);
18
19 @res = SELECT author.ToLowerInvariant() AS author
20     , category
21     , COUNT(*) AS tweetcount
22     FROM @t
23     GROUP BY author.ToLowerInvariant(), category;
24
25 OUTPUT @res
26 TO "/output/MyTwitterAnalysis.csv"
27 ORDER BY tweetcount DESC
28 USING Outputters.Csv();
```

included when you reference the assemblies. The CREATE ASSEMBLY statement syntax grammar looks like this (see the U-SQL language reference documentation at bit.ly/1HWw0cc for more details):

```
Create_Assembly_Statement :=
    'CREATE' 'ASSEMBLY' ['IF' 'NOT' 'EXISTS'] Assembly_Name
    'FROM' Assembly_Source
    ['WITH' "ADDITIONAL_FILES" '='
        '(' Assembly_Additional_File_List ')'];
Assembly_Name := Quoted_or_Unquoted_Identifier.
Assembly_Source :=
    Static_String_Expression | Lexical_binary_value.
```

Speaking of referencing an assembly, U-SQL has a small set of pre-loaded System assemblies and namespaces, including System and System.Linq. The set is kept small to keep the compilation time and the job's resource utilization lower. If you want to refer to a different system assembly, you can just include them with the following statement that adds System.Xml:

```
REFERENCE SYSTEM ASSEMBLY [System.Xml];
```

Once the assembly containing the tweet analysis functions has been registered with the name TweetAnalysis, it can be referenced and used as in **Figure 6**. I need to do a bit more cleanup around the mentions besides just dropping the @ sign; the assembly also contains a cleanup_mentions function that does additional processing beyond dropping the @.

U-SQL Unifies Structured and Unstructured Data

As seen so far, U-SQL makes it very easy to schematize a file on read using the EXTRACT expression. However, once the data preparation has reached a stage where the schema is known, it makes sense to either wrap the EXTRACT into a view or a table-valued function that provides a multi-statement, parameterized view.

Figure 7 shows the new code. The CREATE FUNCTION statement in line 2 creates the U-SQL table-valued function Tweet_Authors_Mentions with the parameter @file that has a provided default value (line 4) and returns the @res rowset of the table type with the three columns author, category and tweetcount (lines 6 to 11). The parameter gets referenced in line 20 and the last assignment to the @res result in line 34 will be returned.

Note that U-SQL table-valued functions are always inlined into the query script, so the U-SQL optimizer can reason and optimize across all the statements. The function can, for example, be called as follows:

```
1 OUTPUT Tweet_Authors_Mentions(DEFAULT)
2 TO "/Samples/Data/Output/MyTwitterAnalysis.csv"
3 ORDER BY tweetcount DESC
4 USING Outputters.Csv();
```

Often, though, the prepared data will be stored as structured data in a U-SQL table that provides additional storage optimizations such as a clustered index and the ability to partition the data. The statements here show how easy U-SQL makes it to create a table by using a CREATE TABLE AS query statement:

```
1 DROP TABLE IF EXISTS TweetAuthorsAndMentions;
2 CREATE TABLE TweetAuthorsAndMentions(INDEX idx
3     CLUSTERED(author ASC)
4     PARTITIONED BY HASH(author) INTO 5
5 )
6 AS Tweet_Authors_Mentions(DEFAULT);
```

Line 3 specifies that the index of the table is clustered by the author column in ascending order and line 4 will horizontally partition the internal representation using a hash on the author values into 5 partitions. After this statement is run the output of the Tweet_Authors_Mentions function will be saved as a new table with

these characteristics. U-SQL also supports round robin and range partitioning for horizontal partitioning schemes, as well as vertical partitioning that allows managing the partitions individually.

Now the table can be used by others to query the data and perform further analysis. The following query in **Figure 8** uses the built-in U-SQL ranking functions together with a windowing expression to calculate the median count of tweets per author and category. It also calculates the relative and absolute ranking position for the author and categories that have more than 50 tweets.

To better understand the SQL-based windowing expressions, let's take a closer look at the expressions in lines 4 to 9. The OVER expression applies the two analytics functions PERCENTILE_DISC and PERCENT_RANK and the ranking function ROW_NUMBER on

Figure 7 Parameterized Table-Valued Function

```

1 DROP FUNCTION IF EXISTS Tweet_Authors_Mentions;
2 CREATE FUNCTION Tweet_Authors_Mentions
3 (
4     @file string = "/Samples/Data/MyTwitterHistory.csv"
5 )
6 RETURNS @res TABLE
7 (
8     author string
9     , category string
10    , tweetcount long?
11 )
12 AS BEGIN
13     REFERENCE ASSEMBLY TweetAnalysis;
14
15     @t =
16         EXTRACT date string
17             , time string
18             , author string
19             , tweet string
20     FROM @file
21     USING Extractors.Csv();
22
23     @m =
24     SELECT AzureConDemo.Udfs.get_ref(tweet) AS refs
25     FROM @t;
26
27     @t =
28     SELECT author, "authored" AS category
29     FROM @t
30     UNION ALL
31     SELECT AzureConDemo.Udfs.cleanup(r) AS r, "referenced" AS category
32     FROM @m CROSS APPLY EXPLODE(refs) AS t(r);
33
34     @res =
35     SELECT author.ToLowerInvariant() AS author
36         , category
37         , COUNT(*) AS tweetcount
38     FROM @t
39     GROUP BY author.ToLowerInvariant(), category;
40 END;
```

Figure 8 Doing Analytics with U-SQL Windowing Expressions

```

1     @res =
2     SELECT DISTINCT
3         author, category, tweetcount
4         , PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY tweetcount ASC)
5         OVER (PARTITION BY category) AS median_tweetcount_perhandle_category
6         , PERCENT_RANK() OVER
7             (PARTITION BY category ORDER BY tweetcount ASC) AS relative_rank
8         , ROW_NUMBER() OVER
9             (PARTITION BY category ORDER BY tweetcount DESC) AS absolute_rank
10    FROM TweetAuthorsAndMentions
11    WHERE tweetcount > 50;
12
13    OUTPUT @res
14    TO "/Output/Demo/tweeter_ranking.csv"
15    ORDER BY absolute_rank, category ASC
16    USING Outputters.Csv();
```

the left-hand side each to a partitioning of the rowset (the so-called windows) specified with the PARTITION BY clause (in all three cases the same partitioning based on the category). The two functions that calculate ranks also need an ordering of data within each window to say where the value is placed. PERCENT_RANK calculates the relative rank of a row within a group of rows specified by the windowing expression. ROW_NUMBER calculates the position of the row within the group of rows. The PERCENTILE_DISC function computes a specific percentile for sorted values in the specified window based on a discrete distribution of the column values. PERCENTILE_DISC(0.5) will compute the 50th percentile (that is, the median) within each window as ordered by the tweet counts in ascending order.

This Is Why U-SQL!

I hope you got a glimpse at why U-SQL makes it easy to query and process Big Data and that you understand the thinking behind the language. The language offers many additional capabilities such as:

- Operates over sets of files with patterns
- Uses vertically partitioned tables
- Federated Queries against Azure SQL DB, SQL Data Warehouse and SQL Server in Azure VMs
- Encapsulates your U-SQL code with Views and Procedures
- More SQL Windowing Functions
- Programs with C# User-defined Operators (custom extractors, processors)
- More Complex Types (MAP, ARRAY)

Please refer to the U-SQL reference documentation for details at bit.ly/1HWw0cc.

To summarize, U-SQL makes Big Data processing easy because it:

- Unifies declarative queries with the expressiveness of your user code
- Unifies querying structured and unstructured data
- Unifies local and remote queries
- Increases productivity and agility from day one

Wrapping Up

U-SQL is just one of the ways Microsoft is working to make Azure Data Lake services the most productive environment for authoring, debugging and optimizing analytics at any scale. With rich support for authoring and monitoring Hive jobs, a C#-based authoring model for building Storm (storm.apache.org) jobs for real-time streaming, and supporting every stage of the job lifecycle from development to operational, Azure Data Lake services lets you focus more on the questions you want to answer than spending time debugging distributed infrastructure. The goal is to make Big Data technology simpler and more accessible to the greatest number possible: Big Data professionals, engineers, data scientists, analysts and application developers. ■

MICHAEL RYS is a principal program manager at Microsoft. He has been doing data processing and query languages since the 1980s. He has represented Microsoft on the XQuery and SQL design committees and has taken SQL Server beyond relational with XML, Geospatial and Semantic Search. Currently he's working on Big Data query languages such as SCOPE and U-SQL when he's not enjoying time with his family underwater or at autocross. Follow him on Twitter: @MikeDoesBigData.

THANKS to the following Microsoft technical experts for reviewing this article:
Omid Afnan and Ed Triou

Real-Time Data Analytics for .NET Developers Using HDInsight

Omid Afnan

Enterprises of all sizes have begun to recognize the value of their huge collections of data—and the need to take advantage of them. As organizations start on their Big Data journey, they usually begin by batch processing their Big Data assets. This could mean collecting and aggregating Web log data, user clicks from an application, telemetry from Internet of Things (IoT) devices or a host of other human- or machine-generated data. I covered the case of doing basic Web log analysis using Hive on HDInsight in an article a year ago (msdn.com/magazine/dn890370). However, as the benefits of batch processing to mine insights on historical data are realized, many organizations encounter the problem of dealing with real-time data and the question of how to collect, analyze and act on continuous streams in real time.

As you might guess, there are technologies in the Big Data space for dealing with such needs. The Microsoft Azure platform provides

powerful Big Data solutions, including Azure Data Lake and HDInsight. There's an open source technology that allows highly distributed real-time analytics called Apache Storm. It's natively supported in HDInsight, which is the Azure managed offering of Apache Big Data services. In this article, I'll walk you through a simple but powerful scenario that deals with a stream of tweets using Storm as the key tool to enable real-time, continuous analytics.

As you'll see, Microsoft makes this kind of development significantly easier than other current market offerings via powerful authoring and debugging tools in Visual Studio. The HDInsight Tools for Visual Studio (available as part of the Azure SDK) provides a coding and debugging environment that's familiar to .NET developers. These tools offer a significantly easier way to work with Big Data technologies than the simple editors and command-line tools currently available in the open source world. While Storm for HDInsight fully supports the use of Java programming, Microsoft also enables .NET programmers to use C# for writing (and reusing) business logic. The examples in this article demonstrate these .NET capabilities.

A Sentiment-Tracking Scenario

The scenario of tracking and analyzing emerging trends is not new. News reporting, weather tracking and disaster detection are examples that pre-date cloud computing. However, both the range of areas where trend detection is desirable, and the scale at which data is available for analysis, have grown unimaginably as the cloud era progresses. Social networking has been fertile ground for

This article discusses:

- Data analytics in the cloud
- Real-time analytics using Apache Storm
- Filtering Tweet streams with .NET
- HDInsight developer tools for Visual Studio

Technologies discussed:

Microsoft Azure, Visual Studio, HDInsight, Apache Storm

Code download available at:

bit.ly/1NG0s23

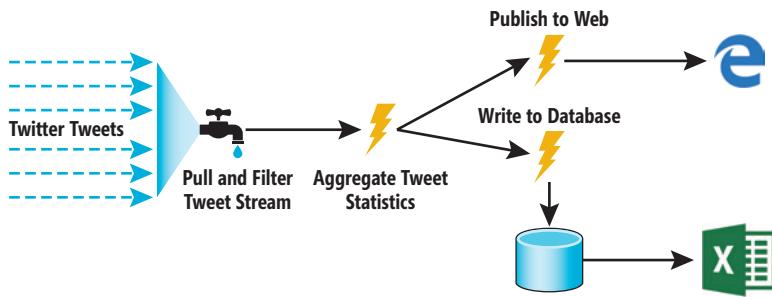


Figure 1 Sentiment Analysis Topology

sentiment analysis. Services like Twitter that make their social data available through APIs, together with pay-as-you-go Big Data platforms like HDInsight, bring sentiment analysis within the reach of organizations large and small.

The simplest kind of sentiment analysis using Twitter is to count how often people are tweeting about a certain topic, or hashtag, in a given period of time. Of course, doing this for just one period, say one minute, is not as interesting as doing this across every minute of the day and looking for a rise or fall in the rate. Identifying spikes in a certain term being used could be useful for detecting a trend. For example, the detection of terms related to a storm or earthquake might provide a very rapid indication of areas affected by a natural disaster and its severity.

To demonstrate the basics of how this is done, I'll walk through how to set up a streaming topology that collects data from Twitter, selects some of the tweets, calculates metrics, saves everything into storage and publishes some of the results. You can see this topology pictured in **Figure 1**. For this article, I selected tweets using simple keyword matching. The metrics calculated are counts of tweets that matched the selection criteria. The selected tweets are put into a SQL database and are also published to a Web site. Everything is done in the Azure cloud using Storm, SQL Server and Web site services available today. After walking through the example, I'll discuss some of the other technologies available for solving this type of streaming data analytics problem.

The Basics of Storm

Storm is an open source Apache project (storm.apache.org) that allows real-time distributed computations to be performed over streams of data. It's part of the Hadoop ecosystem of Big Data processing tools and is directly supported in HDInsight. Storm jobs are defined as a graph of processing nodes connected by streams of data in the form of tuples. Such a graph is referred to as a "topology" in Storm. Topologies don't finish like other queries—they continue to execute until they're suspended or killed.

In the Azure management portal, you can create a new HDInsight cluster and choose Storm as the type. This will cause Azure to set up a cluster of machines preloaded with all the necessary OS, Hadoop and Storm components within minutes. I can choose the number of nodes I want, choose different core and memory sizes, and scale the number of nodes up or down at any time. In terms of simplifying the Hadoop experience, this has already saved me considerable time and headache associated with acquiring and configuring multiple machines.

The components of a topology are called spouts and bolts. Spouts produce streams of tuples, which are basically sets of type and value pairs. In other words, a spout is a piece of code that knows how to collect or generate data and then emit it in chunks. Bolts are units of code that can consume a stream of data. They may process the data to clean it, or calculate statistics. In such cases, they likely emit another stream of tuples to downstream bolts. Other bolts write data to storage or to another system.

Each of these components can execute many parallel tasks. This is the key to the scalability and reliability of Storm. I can specify the degree of parallelism for each component and Storm will allocate that many tasks to execute the logic in my spout or bolt. Storm provides fault tolerance by managing tasks and restarting failed ones automatically. Finally, a given topology executes on a set of worker processes that are essentially execution containers. Workers can be added to increase the processing capacity of a topology. These features provide the essential characteristics that enable scale and fault tolerance for Storm.

A topology can be as complex as necessary to carry out the processing required by the overall real-time analytics scenario. The architecture lends itself to the reuse of components, but it also creates a challenging management and deployment problem as the number of spouts and bolts grows. The Visual Studio project concept is a useful way to manage the code and configuration components needed to instantiate a topology. Because the very idea of a topology is essentially graphical in nature, it also makes sense that being able to visualize the topology is very useful during both development and operation of the system. This can be seen in the execution view of the HDInsight tools for Visual Studio shown in **Figure 2**.

Storm is an open source Apache project that allows real-time distributed computations to be performed over streams of data.

Storm architecture is based on Apache Thrift, a framework that allows the development of services implemented in multiple languages. While many developers use Java to write spouts and bolts, it's not a requirement. With the introduction of the SCP.NET package of libraries, I can use C# to develop my spouts and bolts. This package is included in the HDInsight Tools for Visual Studio download, but can also be downloaded through NuGet.

Filtering Tweets in Near-Real Time

Let's take a look at building the Tweet stream filtering topology to see how these parts work in practice. My sample topology is made up of one spout and three bolts. You can see the graphical view of this topology in **Figure 2**, as shown by the HDInsight Tools for

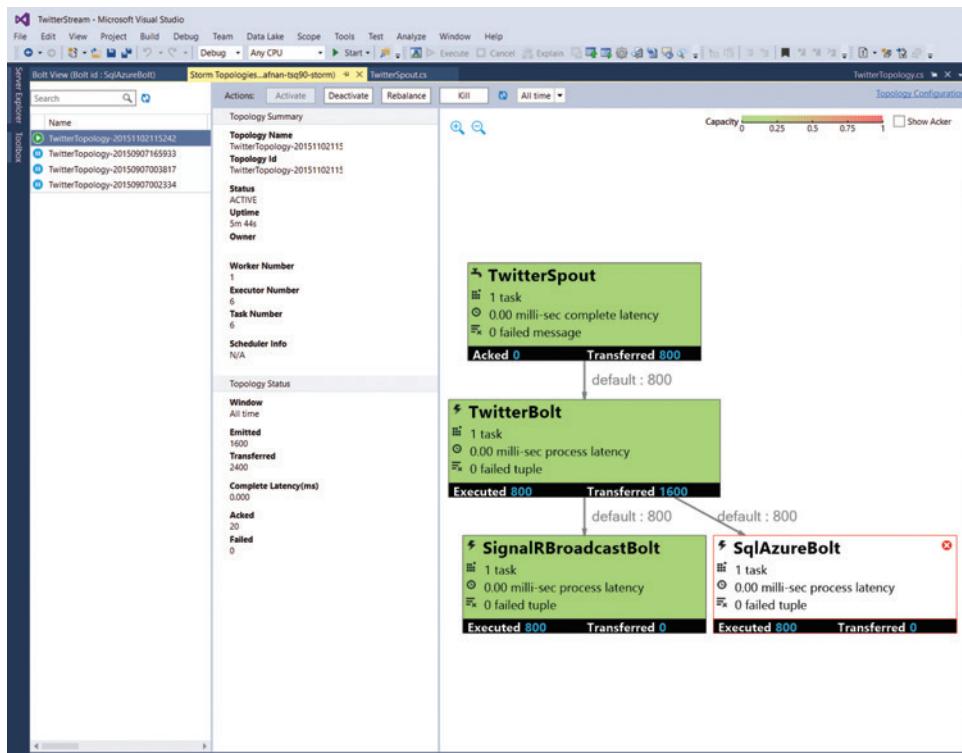


Figure 2 Monitoring View of an Active Storm Topology

Visual Studio. When I submit a Storm project for execution in Azure, Visual Studio shows me this graphical view and updates it over time with the number of events flowing through the system, as well as any error conditions that occur in any of the nodes.

Here TwitterSpout is responsible for pulling the stream of Tweets I want to process. It does so by interacting with the Twitter APIs to collect Tweets, and turns them into tuples of data that can stream through the rest of the topology. TwitterBolt picks up the stream and can do aggregations, like counting Tweets or combining them with other data pulled from other data sources. This bolt emits a new stream, with a possibly new format, based on the business logic it has executed. The AzureSQLBolt and SignalRBroadcastBolt components consume this stream and write portions of the data into an Azure-hosted SQLServer database and a SignalR Web site, respectively.

Because I'm using C# to build my Storm solution, I can use many existing libraries to help simplify and speed up my development. Two key packages for this example are the Tweetinvi libraries on CodePlex (bit.ly/1kl9sqV) and the SCP.NET libraries on NuGet (bit.ly/1QwICPj).

The SCP.NET framework reduces much of the complexity of dealing with the Storm programming model and provides base classes to encapsulate much of the work I'd otherwise need to do by hand. I start by inheriting from the Microsoft.SCP.ISCPSpout base class. This gives me three core methods needed for a spout: NextTuple, Ack and Fail. NextTuple emits the next piece of available data for the stream, or nothing at all. This method is called in a tight loop by Storm, and it's the right place to introduce some sleep time if I don't have any tuples to emit. This is one way to make sure I don't

end up consuming 100 percent of my CPU cycles as the topology runs continuously.

If I want to implement guaranteed message processing, such as "at least once" semantics for my tuples, I'd use the Ack and Fail methods to implement the needed handshakes between bolts. In this example I don't use any retry mechanism, so only the NextTuple method is implemented, using code that takes Tweets from a private queue member in the TwitterSpout class and ships it out to the topology.

Streams within the topology are captured as schemas that are published by a spout or bolt. These are used as the contract between components in the topology, and also as the serialization and de-serialization rules that SCP.NET uses when transferring the data. The Context class is used to store configuration information per instance of a spout or bolt. The

schema of the tuples emitted by the spout is stored in the Context and used by SCP.NET to build component connections.

Let's look at the code for initialization of the TwitterSpout class, shown in part in **Figure 3**.

Figure 3 shows the initialization of the context for this spout using a passed-in context during topology startup. This context is then updated with the addition of a schema definition. I create a Dictionary object in which I add an identifier for the type of stream (DEFAULT_STREAM) and a list of the types for all the fields in

Figure 3 Initializing the TwitterSpout Class

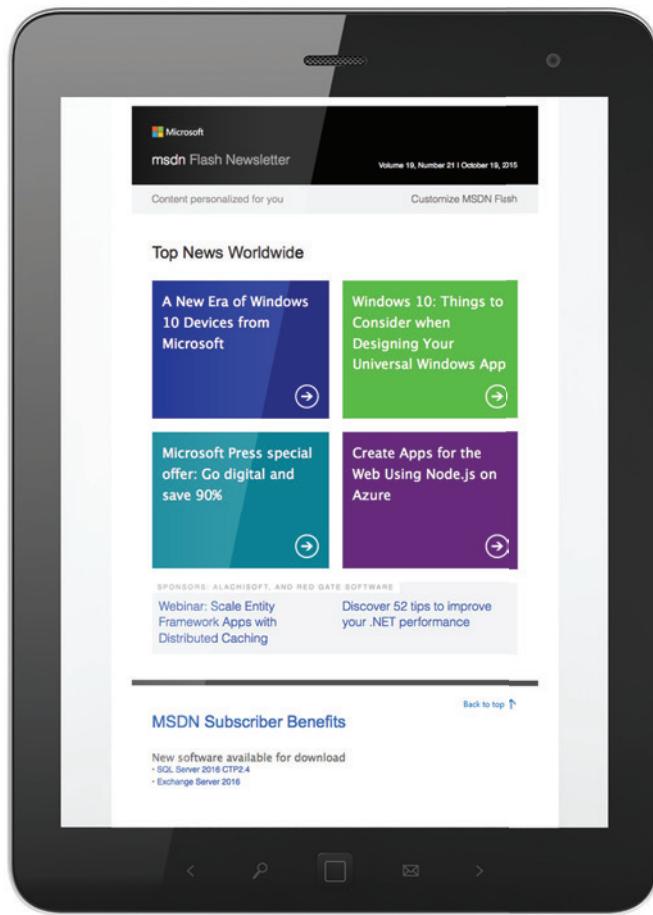
```
public TwitterSpout(Context context)
{
    this.context = context;

    Dictionary<string, List<Type>> outputSchema =
        new Dictionary<string, List<Type>>();
    outputSchema.Add(Constants.DEFAULT_STREAM_ID,
        new List<Type>() { typeof(SerializableTweet) });
    this.context.DeclareComponentSchema(new ComponentStreamSchema(
        null, outputSchema));

    // Specify your Twitter credentials
    TwitterCredentials.SetCredentials(
        ConfigurationManager.AppSettings["TwitterAccessToken"],
        ConfigurationManager.AppSettings["TwitterAccessTokenSecret"],
        ConfigurationManager.AppSettings["TwitterConsumerKey"],
        ConfigurationManager.AppSettings["TwitterConsumerSecret"]);

    // Setup a Twitter Stream
    var stream = Tweetinvi.Stream.CreateFilteredStream();
    stream.MatchingTweetReceived += (sender, args) => { NextTweet(args.Tweet); };

    // Setup your filter criteria
    stream.AddTrack("China");
    stream.StartStreamMatchingAnyConditionAsync();
}
```



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

Figure 4 Specifying Input and Output Schemas for TwitterBolt

```
public TwitterBolt(Context context, Dictionary<string, Object> parms)
{
    this.context = context;
    Dictionary<string, List<Type>> inputSchema =
        new Dictionary<string, List<Type>>();
    inputSchema.Add(Constants.DEFAULT_STREAM_ID,
        new List<Type>() { typeof(SerializableTweet) });
    Dictionary<string, List<Type>> outputSchema =
        new Dictionary<string, List<Type>>();
    outputSchema.Add(Constants.DEFAULT_STREAM_ID,
        new List<Type>() { typeof(long), typeof(string) });
    this.context.DeclareComponentSchema(
        new ComponentStreamSchema(inputSchema,
            outputSchema));
}
```

my tuple—in this case simply a SerializableTweet. The context now contains the schema definition I have to follow when I emit tuples in this class, as well as when I consume them in TwitterBolt.

The rest of this snippet shows the setup of the Twitter stream. The Tweetinvi package provides abstractions for both REST and streaming APIs from Twitter. After encoding the appropriate credentials, I simply instantiate the kind of source I want to use. In the case of streaming sources, I can choose from one of several types, including filtered, sampled or user streams. These provide simplified interfaces for doing keyword filtering across all Tweets, sampling random public Tweets, and tracking events associated with a specific user. Here, I use the filtered stream, which allows the selection of tweets from across all public tweets by checking for the existence of any one of a multiple set of keywords.

Here I perform the desired filtering of Tweets at the spout, because the Tweetinvi APIs make this easy to do. I could also do filtering in the TwitterBolt component, along with any other calculations or aggregations I want to do to massage the Tweets. Filtering at the spout allows me to reduce the volume of data streaming through the topology at an early stage. However, the power of Storm is that it allows me to handle large volumes at any component in the topology by scaling out. Storm provides almost linear scaling with added resources, which allows me to use more workers to add scale wherever a bottleneck occurs. HDInsight supports this approach by letting me select the size of my cluster and the types of nodes when I set it up, and to add nodes to it later. Using this scale-out approach, I can build Storm clusters that process millions of events per second. I'm charged by the number of running nodes in my cluster, so I have to keep in mind the trade-off between cost and scale.

The only other part to call out in **Figure 3** is the registration of a callback for the Tweetinvi stream object to call when it finds a tweet that matches my criteria. The NextTweet method is that callback, which simply adds the provided tweet to the previously mentioned private queue in the TwitterSpout class:

```
public void NextTweet(ITweet tweet)
{
    queue.Enqueue(new SerializableTweet(tweet));
}
```

The bolts for my topology are coded in a similar fashion. They derive from the Microsoft.SCPCBolt class and must implement the Execute method. Here the tuple is passed in as a generic type of SCPTuple and must be converted to the correct type first. Then I can write C# code to execute whatever detailed processing

I need. In this case I simply use a global variable to accumulate a count of the number of tuples seen by the bolt, and log the count and tweet text. Finally, I emit a new type of tuple for downstream bolts to consume. Here's the code:

```
public void Execute(SCPTuple tuple)
{
    var tweet = tuple.GetValue(0) as SerializableTweet;
    count++;
    Context.Logger.Info("ExecuteTweet: Count = {0}, Tweet = {1}", count, tweet.Text);
    this.context.Emit(new Values(count, tweet.Text));
}
```

In the case of a bolt, I have to specify both input and output schemas when I set it up. The format is exactly the same as the previous schema definition for a spout. I simply define another Dictionary variable called outputSchema and list the integer and string types of the output fields, as shown in **Figure 4**.

The other bolts follow the same pattern, but call specific APIs for SQL Azure and SignalR. The final key element is to define the topology by enumerating the components and their connections. To accomplish this, there's another method that must be implemented in all spouts and bolts—the Get method, which simply instantiates an object of this class with a Context variable that gets called by the SCPContext during the launch of the Storm task. SCP.NET will instantiate a child C# process, which will start your C# spout or bolt task using the following delegate method:

```
return new TwitterSpout(context);
```

Figure 5 Creating the Twitter Analysis Topology

```
namespace TwitterStream
{
    [Active(true)]
    class TwitterTopology : TopologyDescriptor
    {
        public ITopologyBuilder GetTopologyBuilder()
        {
            TopologyBuilder topologyBuilder = new TopologyBuilder()
                .Typeof(TwitterTopology).Name + DateTime.Now.ToString("-yyyyMMddHHmmss"));

            topologyBuilder.SetSpout(
                Typeof(TwitterSpout).Name,
                TwitterSpout.Get,
                new Dictionary<string, List<string>>()
            {
                {Constants.DEFAULT_STREAM_ID, new List<string>() {"tweet"}}
            },
            1);

            topologyBuilder.SetBolt(
                Typeof(TwitterBolt).Name,
                TwitterBolt.Get,
                new Dictionary<string, List<string>>()
            {
                {Constants.DEFAULT_STREAM_ID, new List<string>() {"count", "tweet"}}
            },
            1).shuffleGrouping(Typeof(TwitterSpout).Name);

            topologyBuilder.SetBolt(
                Typeof(SqlAzureBolt).Name,
                SqlAzureBolt.Get,
                new Dictionary<string, List<string>>(),
                1).shuffleGrouping(Typeof(TwitterBolt).Name);

            topologyBuilder.SetBolt(
                Typeof(SignalRBroadcastBolt).Name,
                SignalRBroadcastBolt.Get,
                new Dictionary<string, List<string>>(),
                1).shuffleGrouping(Typeof(TwitterBolt).Name);
        }
    }
}
```

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

CODE

Las Vegas

WE CAN BELIEVE IN

MARCH 7-11, 2016

BALLY'S HOTEL & CASINO



Visual Studio
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE!

DEVELOPMENT TOPICS INCLUDE:

- ALM / DevOps
- ASP.NET
- Cloud Computing
- Database & Analytics
- JavaScript / HTML5 Client
- Mobile Client
- Modern App Development
- UX / Design
- Visual Studio / .NET
- Windows Client

BONUS
CONTENT!

Presented in
Partnership with
Magenic

Modern Apps **LIVE!**

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

REGISTER BY JANUARY 20
AND SAVE \$400!

USE PROMO CODE VSLJANTI

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



VSLIVE.COM/VEGAS

JOIN US

ON THE **2016**
CAMPAIGN FOR CODE
TRAIL!



March 7-11



May 16-19



June 13-16



August 8-12



September 26-29



October 3-6



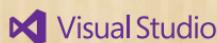
December 5-9



SUPPORTED BY



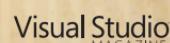
Microsoft



Visual Studio



msdn
magazine



Visual Studio
MAGAZINE

PRODUCED BY



1105 MEDIA[®]
YOUR GROWTH. OUR BUSINESS.

With the spouts and bolts in place, I can now create the topology. Again, SCP.NET provides a class and helper functions to do this. I create a class derived from Microsoft.SCP.Topology.TopologyDescriptor and override the GetTopologyBuilder method. In this method I use an object of type TopologyBuilder that provides the methods SetSpout and SetBolt. These methods let me specify the name and input and output schemas of the component. They also let me specify the Get delegate to use to initialize the component and, most important, to specify the upstream component to connect with the current component. **Figure 5** shows the code defining my topology.

The complete Twitter analysis project can be built in Visual Studio using the Storm project type. This project conveniently lays out the various components you need in a simple and familiar way that can be viewed in Solution Explorer, as shown in **Figure 6**. You can add components like bolts and spouts using the Add | New Item option from the context menu of a project. Choosing from the Storm item types adds a new file and includes the outline for all the required methods. Using the Visual Studio Storm project, I can add references to libraries, like Tweetinvi, directly or through NuGet. Submitting the topology to run on Azure is a single click from the Solution Explorer context menu. All necessary components are uploaded to the HDInsight Storm cluster of my choice and the topology is submitted.

After submission, I see the Topology View from **Figure 2**, where I can monitor the state of my topology. Storm allows several states for my topology, including activated, deactivated and killed, as well as allowing a rebalance of tasks across workers based on scalability parameters. I can manage all of these state transitions from Visual Studio, as well as observe the current flow of tuples. In order to

investigate components in detail and debug issues, I can drill down into individual components like the SqlAzureBolt, which is showing an error condition (the red outline and marker on the Topology View). Double-clicking this bolt shows more detailed stats about tuple flow, as well as the summary of errors from the bolt. You can even click on the Error Port link to go to the full logs of individual tasks without having to leave Visual Studio.

The code and the project for the simple topology covered in this article can be found on GitHub under the MicrosoftBigData repository. Look for the HDInsight folder and the TwitterStream sample project. You'll find additional articles and samples at bit.ly/1MCfqsM.

Moving to More Complex Analysis

The Storm topology example I presented is a simple one. There are a number of ways I can increase the power and complexity of my real-time processing in Storm.

As already mentioned, the number of resources assigned to a Storm cluster in HDInsight can be scaled up as needed. I can observe the performance of my system from the data provided in Visual Studio's runtime view of the topology from **Figure 2**. Here I can see the number of tuples being emitted, number of executors and tasks and latencies. **Figure 7** shows the Azure Management Portal view, which provides further details about the number of nodes, their type and core counts that are in use now. Based on these, I can decide to scale my cluster and add more supervisor (worker) nodes to the cluster. This scale-up doesn't require a restart and will happen within minutes when I trigger a rebalance from my Visual Studio topology view or from the Management Portal.

Most analytics applications will operate on multiple unstructured Big Data streams. In this case, the topology would contain multiple

spouts and bolts that can read from more than one spout. This can be expressed easily in the topology configuration by specifying several inputs in the SetBolt method invocation. However, the business logic of dealing with the multiple sources in the same bolt will be more complex as individual tuples arrive under different stream IDs. As the complexity of the business problem grows, it's likely that relational or structured data sources will also be needed during processing. While spouts are ideal for queue-like data sources, relational data is more likely to be brought in by a bolt. Again, the flexible implementation of bolts and the use of C# or Java make it possible to easily code access to a database using established APIs or query languages. The complexity here arises from the fact that these calls will be made remotely from Storm containers in

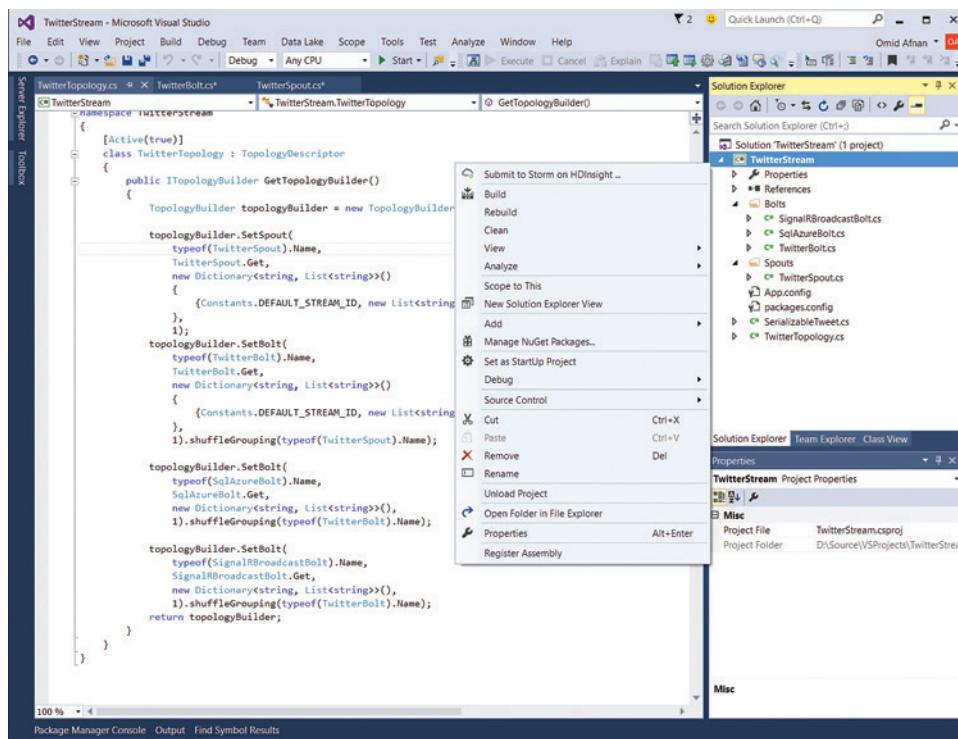


Figure 6 Submitting a Topology from Solution Explorer

a cluster to the database server. SQL Azure and HDInsight work on the same Azure fabric and interact easily, but there are other choices for cloud-based services that can also be used.

The Storm runtime allows me to set or tweak many fine-grain behaviors of the system. Many such settings appear as configuration parameters that can be applied at the topology or task level. These can be accessed in the Microsoft.SCP.Topology.StormConfig class and used to tune the overall Storm workload. Examples include settings for the maximum number of pending tuples per spout, tick tuples and spout sleep strategy. Other changes to the topology can be made in the topology builder. In my example topology, the streaming between all components is set to “shuffle grouping.” For any given component, the Storm execution system can and will create many individual tasks. These tasks are independent worker threads that can run in parallel across cores or containers to spread the workload for the bolt across multiple resources. I can control how work is passed from one bolt to the next. By choosing shuffle grouping, I’m saying that any tuple can go to any worker process in the next bolt. I can also choose other options like “field grouping,” which would cause tuples to be sent to the same worker based on the value of a particular field in the tuple. This option can be used to control the flow of data for operations that have a state, like a running count for a particular word in the Tweet stream.

Finally, a real-time analytics system might be part of a larger pipeline of analytics within an organization. For example, a Web log analytics system is likely to have a large batch-oriented portion that processes the logs for a Web service on a daily basis. This would produce Web site traffic summaries and provide lightly aggregated data suitable for pattern discovery by a data scientist. Based on this analysis, the team might decide to create real-time triggers for

certain behaviors, like detection of system failures or malicious use. This latter portion would require real-time analysis of log or telemetry streams, but is likely to depend on reference data that’s updated daily by the batch system. Such larger pipelines require a workflow management tool that allows the synchronization of tasks across a variety of computation models and technologies. The Azure Data Factory (ADF) provides a workflow management system that natively supports the Azure analytics and storage services and allows coordination across tasks based on input data availability. ADF supports HDInsight and Azure Data Lake Analytics, as well as moving data between Azure Storage, Azure Data Lake Storage, Azure SQL Database and on-premises data sources.

Other Streaming Technologies

In this article I introduced the basics of real-time streaming analytics using Storm in HDInsight. Of course, Storm can also be set up on your own cluster of machines in your own datacenter or lab. The Storm distribution can be obtained through Hortonworks, Cloudera or directly from Apache. The installation and configuration in these cases is considerably more time-consuming, but the concepts and code artifacts are the same.

Spark (spark.apache.org) is another Apache project that can be used for real-time analytics and has gained great popularity. It supports general Big Data processing, but its support for in-memory processing and a library of streaming functions makes it an interesting choice for high-performance real-time processing. HDInsight offers Spark cluster types where you can experiment with this technology. The service includes Zeppelin and Jupyter notebooks, which are interfaces that let you build queries in these languages and see interactive results. These are ideal for data exploration and developing queries over Big Data sets.

The interest in real-time streaming analytics is building as organizations work their way through increasingly complex scenarios for Big Data analytics. At the same time, technologies in this space continue to grow and mature, providing new opportunities for gaining insights from Big Data. Look to these pages for future articles on the use of technologies such as Spark and Azure Data Lake Analytics. ■

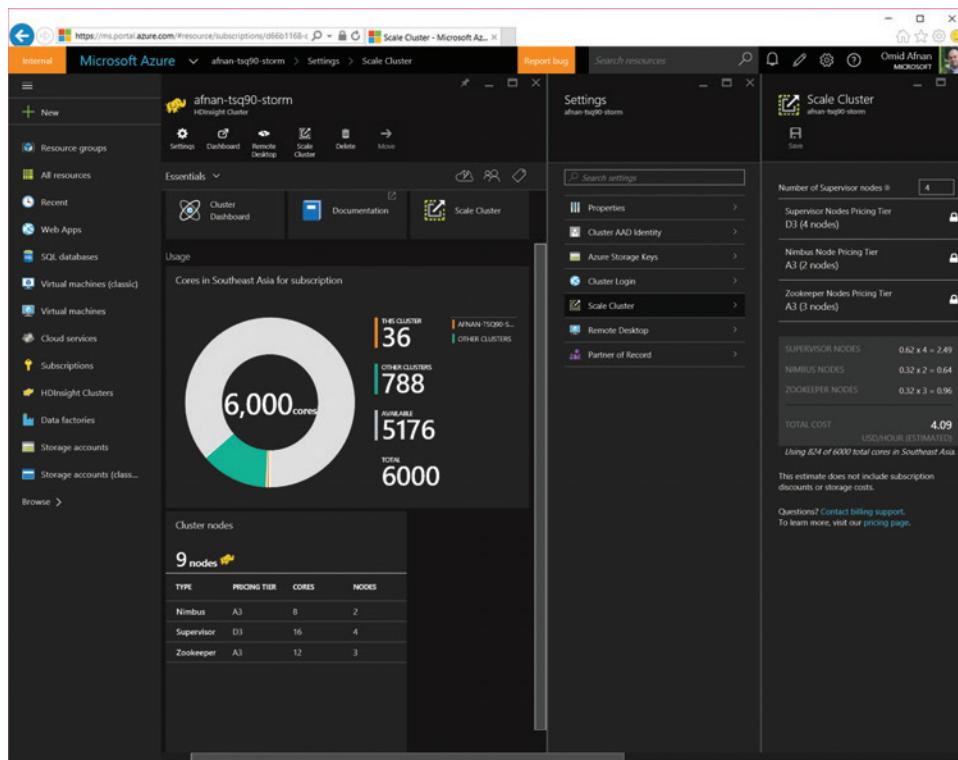


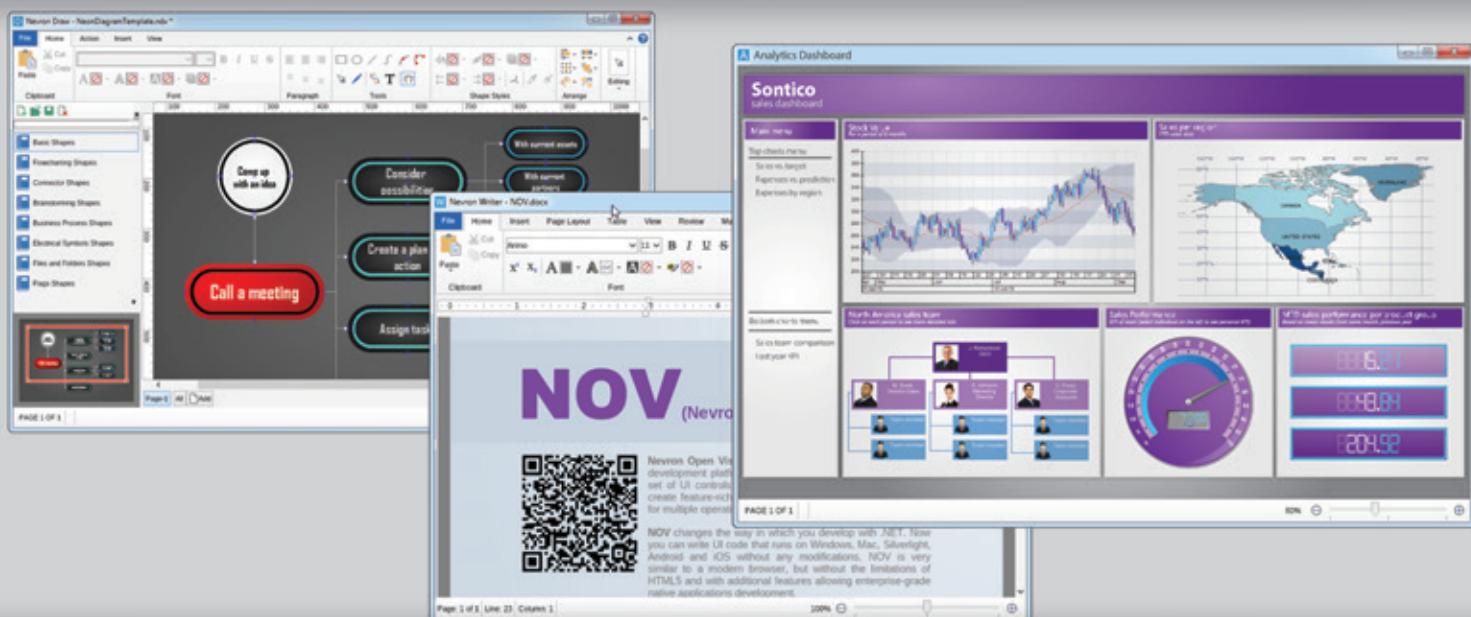
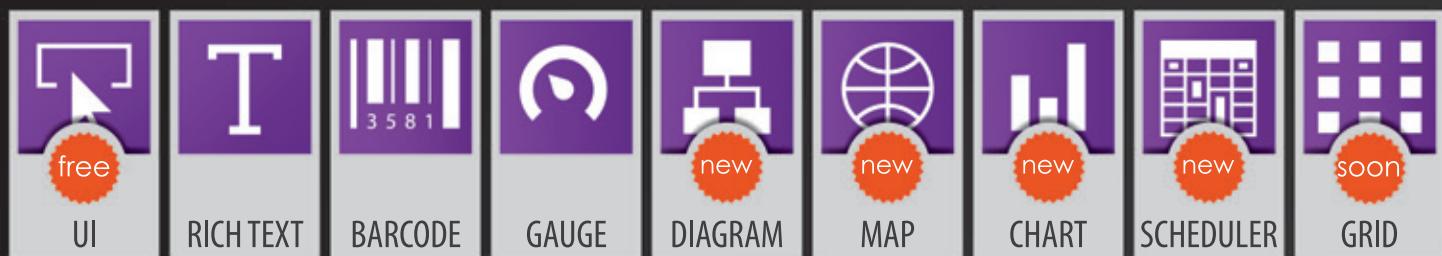
Figure 7 Azure Management Portal View of Storm Cluster

OMID AFNAN is a principal program manager in the Azure Big Data team working on implementations of distributed computation systems and related developer toolchains. He lives and works in China. Reach him at omafnan@microsoft.com.

THANKS to the following technical experts for reviewing this article:
Asad Khan and Ravi Tandon



The Complete UI suite for .NET



develop cutting edge applications for WinForms, WPF, Silverlight, Xamarin.Mac & MonoMac using a single code base

DOWNLOAD FREE TRIAL

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

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

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

Creating Big Data Pipelines Using Azure Data Lake and Azure Data Factory

Gaurav Malhotra

This year Microsoft Azure Big Data offerings were expanded when the Azure Data Lake (ADL) service, along with the ability to create end-to-end (E2E) Big Data pipelines using ADL and Azure Data Factory (ADF) were announced. In this article, I'll highlight the use of ADF to schedule both one-time and repeating tasks for moving and analyzing Big Data.

ADL makes processing Big Data simpler and more accessible by providing several key technologies. The U-SQL language is a powerful combination of SQL and C# that supports parallel execution. You can run U-SQL in the ADL Analytics cloud offering, where hundreds or thousands of containers can be reserved, used and released in the lifetime of a single job. Setting up this cloud environment with ADL is easy. Using the Azure Management portal, you can quickly and easily create ADL accounts for both storage and analytics, and provision an ADF. Within minutes, with

This article discusses:

- A scenario for analyzing Web logs using Azure Data Lake and Azure Data Factory
- Moving data to Azure Data Lake Store
- Creating a pipeline with U-SQL activities
- Monitoring Big Data pipelines

Technologies discussed:

Azure Data Lake, Azure Data Factory, U-SQL

the click of a few buttons, you can set up all the necessary accounts in your Azure subscription.

Once the account provisioning is complete, you can create end-to-end Big Data pipelines in ADF using the Azure Management Portal, Windows PowerShell, the C# SDK and Visual Studio tools. ADF is a cloud-based data integration service that orchestrates and automates the movement and transformation of data. The ADF-ADL integration allows you to:

- Move data from a given source to the ADL Store.
- Create Big Data ADF pipelines that run U-SQL as a processing step on the ADL Analytics service.

There are a number of common Big Data scenarios that ADL and ADF address, including customer churn analysis, personalized product recommendations and actuarial processing. One that's interesting to many Azure customers is analyzing Web services or application logs. In this article, I'll show you how you can create data factory pipelines to analyze Web logs, by first moving your Web logs to ADL and then running U-SQL scripts to process them.

The Web Log Analysis Scenario

A common business insight scenario is the analysis of Web logs to understand the volume and pattern of user requests based on the originating regions or locales around the world. Such analysis enhances customer understanding, marketing campaigns and product roadmaps, including localization plans. Logs are emitted by Web applications, network devices, OSes, and all manner of

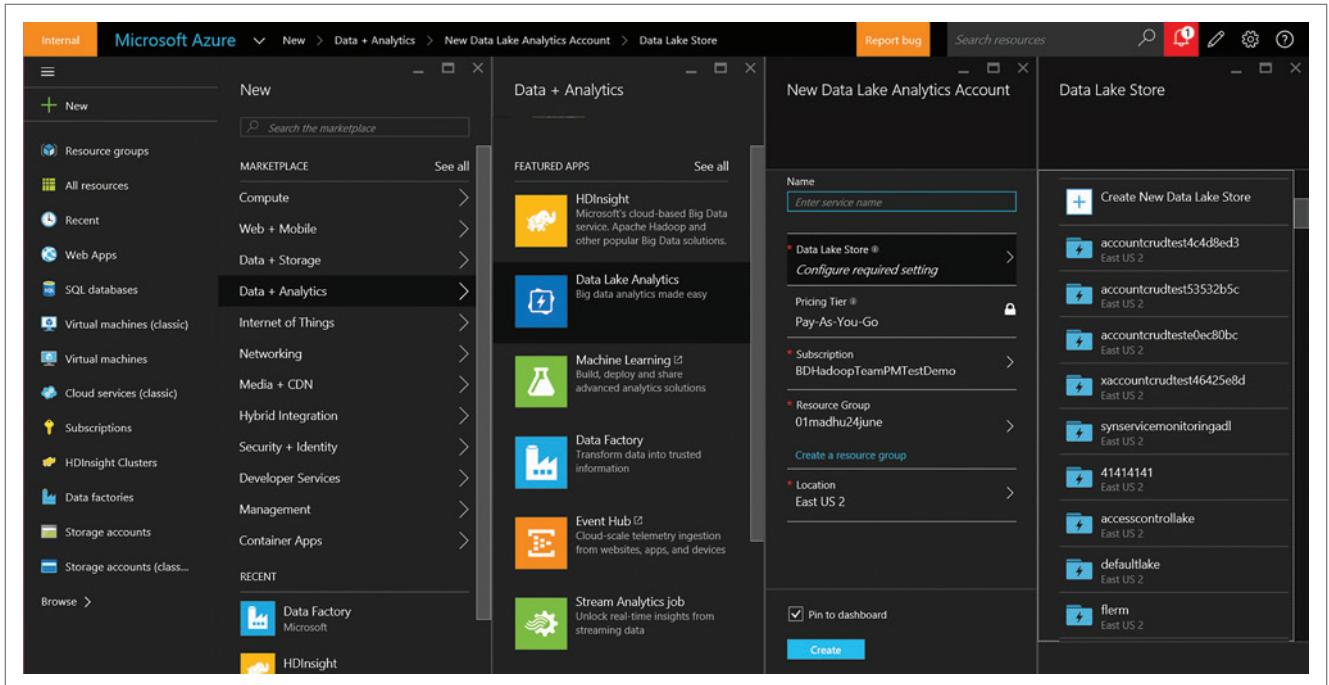


Figure 1 Creating ADL Accounts in the Azure Management Portal

intelligent or programmable devices. Data streams, such as error logs, clickstream instrumentation and Web server logs, can easily accumulate at the rate of gigabytes or terabytes a week. The Web logs can accumulate in any form of storage, including SQL Azure, Azure Blob Storage, Amazon Simple Storage Service (S3), and on-premises and Oracle databases. Analyzing these logs quickly and efficiently to discover usage patterns and system issues helps companies understand customer usage better and, ultimately, drive up customer engagement and satisfaction.

Data streams, such as error logs, clickstream instrumentation and Web server logs, can easily accumulate at the rate of gigabytes or terabytes a week.

It's easy to store these Web logs in the ADL Store, which can elastically scale to store petabytes of data. When provisioning an ADL Store account, no sizing parameters are needed—the account simply grows to accommodate the size of the files loaded into it. Because you pay only for what's actually stored, the service is cost-effective, and the lack of fixed limits on account or file size, plus massive throughput capability, make it ideal for performing Big Data analytics. In addition, you don't have to rewrite code or move your data to a different storage platform as the size of data stored increases or decreases.

Parsing and aggregating Web logs based on certain partitions, like region, are activities that allow a high degree of parallelism. The ideal case is to have subsets of records farmed out to individual servers that can parse, transform and summarize those records. Then, these partial results are merged in multiple parallel stages until the final aggregated dataset is created. Managing this process manually is extremely complicated and prone to sub-optimal execution based on incomplete information about the system and the changing shape of the data from day to day. However, this is exactly what ADL Analytics and the U-SQL language do automatically. U-SQL allows you to express your target aggregations in a declarative SQL-like query syntax that doesn't require specifying any parallelism directives. The compiler and scheduler then figure out the degree of parallelism inherent in the job and allocate resources based on that parallelism and any limits specified for maximum resource usage. Using ADF, you can easily build up a pipeline by specifying such U-SQL tasks; connect them with other series of tasks; add activities to move data from your Web servers to ADL Store; and create a schedule to regularly process the data. The simplicity of creating the pipeline and components lets you focus on your business logic instead of on how to optimize processing and storing large datasets.

Getting Set up

You start by creating the ADL Store and analytics accounts and provisioning a data factory. This is all done through the Azure Management Portal. An ADL analytics account is the entity that helps you group and manage the queries and programs you run to do Big Data analysis. You can manage billing by associating your account with different Azure subscriptions, and choose pricing tiers. There are options for grouping the account with other resources for tracking purposes. You can even choose the region

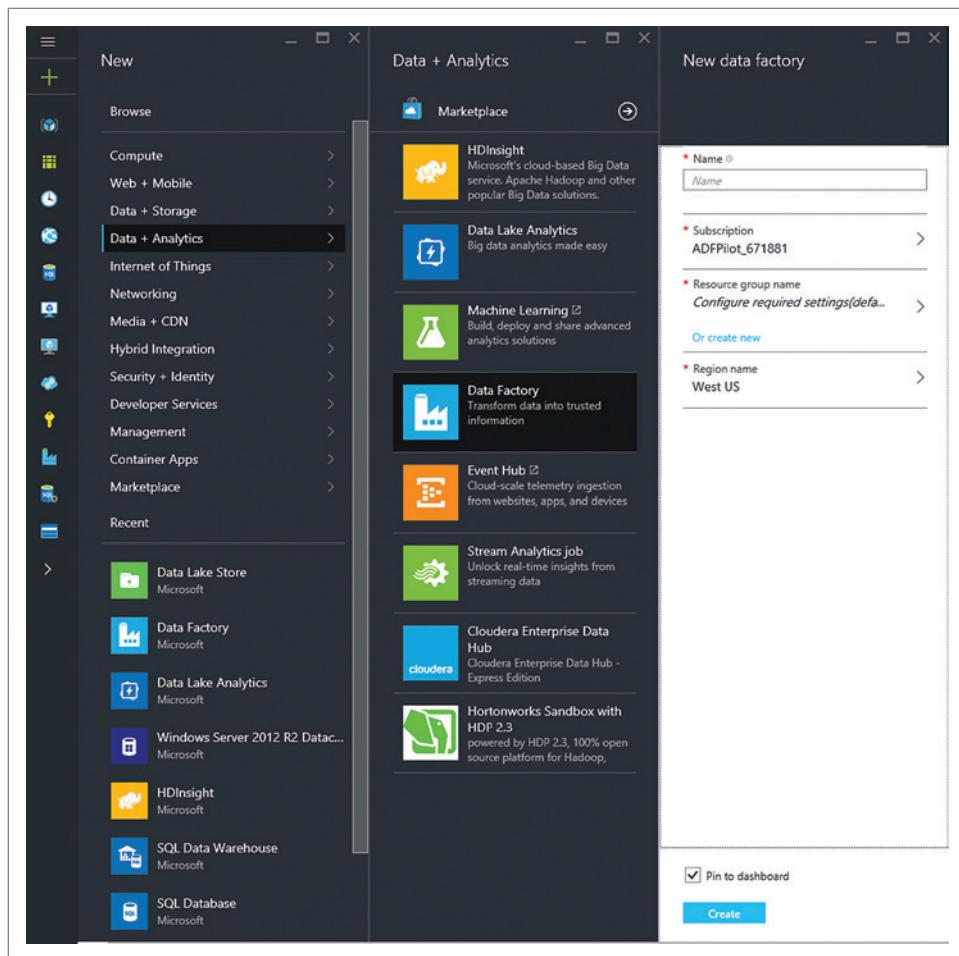


Figure 2 Provisioning an Azure Data Factory

of the datacenter where your account lives, which can be useful for managing proximity to on-premises data.

The ADL Store service is a service for storage of Big Data that can be accessed from HDFS-compliant systems, including business intelligence (BI) tools and on-premises applications. The setup is very simple and you don't have to specify any limits at setup time. As with the analytics service, an important option is the geographic region where you want the data housed. In the case of data storage, this can be critical as there might be business requirements related to legal compliance for where data about a region's citizens is stored. ADL Store accounts can be created separately and used with other services, but the most common case is to create an account in tandem with ADL Analytics. **Figure 1** shows the screen for creating an ADL account. You provide a name for the account and choose your subscription, resource group and location. The Create New Data Lake Store option lets you

activities into a unit that together performs a task. You'll see this in more detail as I walk through setting up the data factory for the weblog analytics.

create a new store at the same time and with the same options as your Analytics account.

In this example, the Web log data is stored in Azure Storage, which allows you to store blob data on Azure. You can also create Azure Storage through the portal. Whether you use an Azure Web App, or a Web site hosted somewhere else, getting your data onto Azure Storage is easy and means it will have high availability and durability. The sample Web log data used in the example can be found at bit.ly/10Ni8c5.

Figure 2 shows how you can provision an Azure data factory. As you can see, the process of setting up these cloud-based services is very easy. You don't need to use the same geographic region as the ADL services—the data factory can operate with services in any region.

Data factories are a combination of data stores, linked services and pipelines. Data stores and linked services are definitions of external entities that usually already exist outside of ADF. Pipelines are a logical grouping of activities in ADF. They are used to group

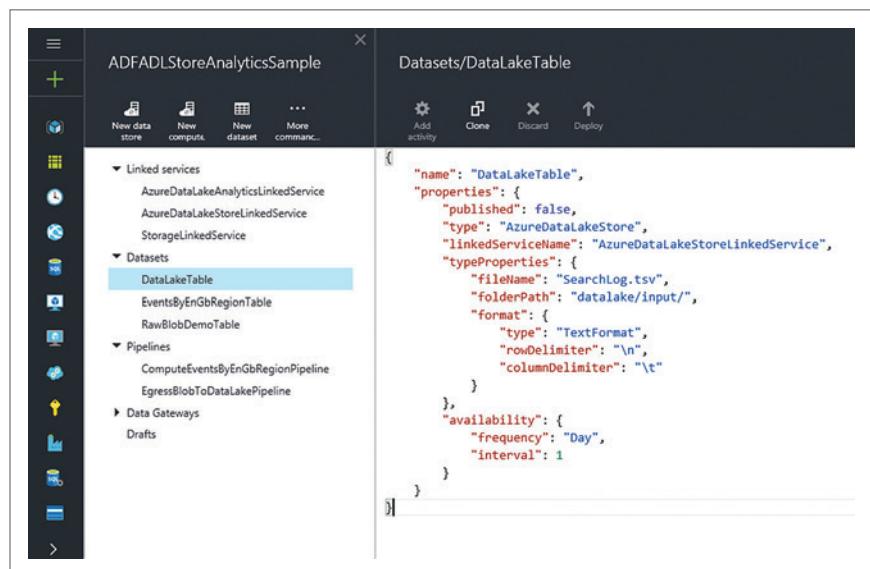


Figure 3 Author and Deploy a Data Factory Using the Web Editor



NEW Universal HTML5 Viewer & Document Management Kit



zero-footprint solution



super-easy integration



fast & crystal clear rendering



fully customizable UI
look & feel



mobile devices optimization



annotations, thumbnails
bookmarks & text search

supports nearly 100 formats

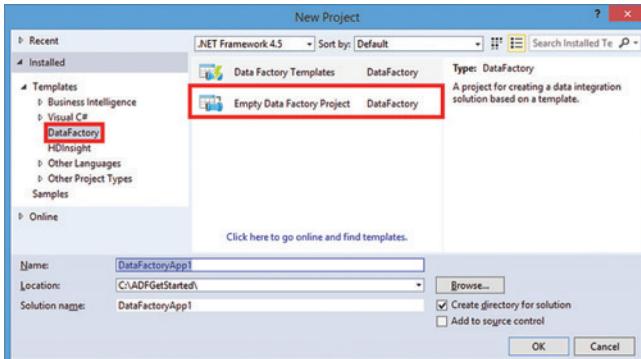


Figure 4 Author and Deploy a Data Factory Using the Visual Studio Plug-In

You can create these entities using the Azure Management Portal or Visual Studio. In the portal, under the Data Factory view, an Author and Deploy option allows you to select the individual components of a data factory by type, and provides JSON snippets that can be edited directly and published (see **Figure 3**). Alternatively, you can take advantage of the ADF tools for Visual Studio and use a project format to identify and define each of the components of the data factory (**Figure 4**). The project can then also be published to create these entities in your data factory in Azure.

Moving Data to Azure Data Lake Store

The first step in the Web log analysis scenario is to move the data to ADL Store. You can move data to ADL Store using the Copy activity in an ADF pipeline. In order to do the copy operation, you need to create ADF linked services, datasets and pipelines. Linked services in ADF define the information needed to connect to external resources. Linked services are used for two purposes in Data Factory. The first is to represent a data store including, but not limited to, an on-premises SQL Server, Oracle database, file share or Azure blob storage account. The second is to represent a processing resource that can host the execution of an activity. For example, the HDInsight Hive activity executes on an HDInsight Hadoop cluster. In this case, you need to create two linked services, one corresponding to the Azure Storage account and the second representing the ADL Store.

The Copy activity in ADF is very powerful and allows you to copy data between on-premises or cloud sources and sinks that may have different schemas.

You also need to create two ADF datasets. Datasets are logical references to data in an Azure Storage account or ADL Store. No user data is stored in ADF itself, so dataset definitions are needed for ADF to identify the structure of data in the external data stores, including tables, files, folders and documents. Because ADF doesn't

know the structure of this data, you need to define it here so the system knows what columns and types to expect. In this case, you need to create one dataset corresponding to the Azure Storage account location that contains the Web log data (source) and a second dataset corresponding to the ADL Store where you want to move the Web logs (sink).

Finally, for the data copy to happen, you need to create an ADF pipeline that contains a Copy activity. An ADF pipeline is a logical grouping of activities, such as data Copy, that can run at different intervals, and Hive, Pig, or U-SQL script activities that can run regularly—every 15 minutes, hourly, daily, or monthly. The Copy activity in ADF is very powerful and allows you to copy data between on-premises or cloud sources and sinks that may have different schemas. You can specify a few parameters, or accept defaults, to begin. You have a lot of control and can tune things like the schedule and policies for handling error conditions.

Although a pipeline can run on a repeating schedule, in the current example it's run just once to move the data into the ADL Store. The JSON snippet in **Figure 5** shows a definition for a pipeline called EgressBlobToDataLakePipeline. This pipeline has a Copy activity to

Figure 5 EgressBlobToDataLakePipeline Sample Pipeline Definition

```
{
  "name": "EgressBlobToDataLakePipeline",
  "properties": {
    "description": "Egress data from blob to azure data lake",
    "activities": [
      {
        "type": "Copy",
        "typeProperties": {
          "source": {
            "type": "BlobSource",
            "treatEmptyAsNull": true
          },
          "sink": {
            "type": "AzureDataLakeStoreSink",
            "writeBatchSize": 10000,
            "writeBatchTimeout": "00:10:00"
          }
        },
        "inputs": [
          {
            "name": "RawBlobDemoTable"
          }
        ],
        "outputs": [
          {
            "name": "DataLakeTable"
          }
        ],
        "policy": {
          "timeout": "10:00:00",
          "concurrency": 1,
          "executionPriorityOrder": "NewestFirst",
          "retry": 1
        },
        "scheduler": {
          "frequency": "Day",
          "interval": 1
        },
        "name": "EgressDataLake",
        "description": "Move data from blob to azure data lake"
      }
    ],
    "start": "2015-08-08T00:00:00Z",
    "end": "2015-08-08T01:00:00Z",
    "isPaused": false
  }
}
```

move data from Azure Blob Storage to Azure Data Lake Store. It's scheduled to run on 08/08/2015 and will only run once (the "start" and "end" properties for the pipeline active period are the same).

When the copy activity in the ADF pipeline completes successfully, the Web logs have been moved from Azure Blob Storage to Azure Data Lake Store. You can learn more about Azure Data Factory data movement activities at bit.ly/1MNblqZ, and more about using AzureDataLakeStore connector in ADF at bit.ly/1MRwWZ. Now you're ready to process and analyze Web logs.

Creating a Pipeline with U-SQL Activities

With the data in ADL Store, you can now run U-SQL scripts on ADL Analytics service to process and analyze the Web logs. You can create pipelines that will consume the data from ADL Store, run the U-SQL scripts on ADL Analytics service as a processing step and produce the output in ADL Store. The downstream applications can then consume the processed output directly from ADL Store or you can choose to copy the data from ADL Store to Azure SQL Data warehouse if your BI applications are using a SQL warehouse as a back-end store.

Figure 6 ComputeEventsByEnGbRegionPipeline Sample Pipeline Definition

```
{  
  "name": "ComputeEventsByEnGbRegionPipeline",  
  "properties": {  
    "description": "This is a pipeline to compute events for en-gb locale  
      and date less than 2012/02/19.",  
    "activities": [  
      {  
        "type": "DataLakeAnalyticsU-SQL",  
        "typeProperties": {  
          "scriptPath": "scripts\\usql\\SearchLogProcessing.txt",  
          "scriptLinkedService": "StorageLinkedService",  
          "degreeOfParallelism": 3,  
          "priority": 100,  
          "parameters": {  
            "in": "/datalake/input/SearchLog.tsv",  
            "out": "/datalake/output/Result.tsv"  
          }  
        },  
        "inputs": [  
          {  
            "name": "DataLakeTable"  
          }  
        ],  
        "outputs": [  
          {  
            "name": "EventsByEnGbRegionTable"  
          }  
        ],  
        "policy": {  
          "timeout": "06:00:00",  
          "concurrency": 1,  
          "executionPriorityOrder": "NewestFirst",  
          "retry": 1  
        },  
        "scheduler": {  
          "frequency": "Day",  
          "interval": 1  
        },  
        "name": "EventsByRegion",  
        "linkedServiceName": "AzureDataLakeAnalyticsLinkedService"  
      }  
    ],  
    "start": "2015-08-08T00:00:00Z",  
    "end": "2015-08-08T01:00:00Z",  
    "isPaused": false  
  }  
}
```

In order to process the Web logs, you need to create ADF linked services, datasets and pipelines again. You can reuse the ADL Store linked service created in the previous step if you want to create a sequence of pipelines that do the data movement first and then perform data analysis by running U-SQL scripts in a single ADF. Or, you can create a new data factory that just performs the data analysis. The ADF pipeline in this case contains an Azure Data Analytics U-SQL activity and runs a U-SQL script to determine all events for the Great Britain ("en-gb") locale and a date less than "2012/02/19." **Figure 6** contains the JSON definition for ComputeEventsByEnGbRegionPipeline, which defines such a pipeline with a U-SQL activity to do Web log processing.

I can create pipelines that will consume the data from ADL Store, run the U-SQL scripts on ADL Analytics service as a processing step and produce the output in ADL Store.

The U-SQL script in **Figure 7** being run by the pipeline resides in the scripts/usql folder (the scriptPathproperty in the pipeline JSON in **Figure 5**) in the Azure Blob Storage account corresponding to the deployed StorageLinkedService. The values for @in and @out parameters in the script are passed dynamically by ADF using the Parameters section in the pipeline JSON (see the "Parameters" section in **Figure 6**). You can also specify other properties, such as degreeOfParallelism or priority in your pipeline definition for the jobs that run on the ADL Analytics service. This U-SQL script processes Web logs and returns all events for the "en-gb" locale and date less than "2012/02/19."

Figure 7 The U-SQL Script SearchLogProcessing.txt

```
@searchlog =  
  EXTRACT UserId      int,  
          Start       DateTime,  
          Region     string,  
          Query      string,  
          Duration   int?,  
         Urls       string,  
          ClickedUrls string  
  FROM @in  
  USING Extractors.Tsv(nullEscape:"#NULL#");  
  
@rs1 =  
  SELECT Start, Region, Duration  
  FROM @searchlog  
  WHERE Region == "en-gb";  
  
@rs1 =  
  SELECT Start, Region, Duration  
  FROM @rs1  
  WHERE Start <= DateTime.Parse("2012/02/19");  
  
OUTPUT @rs1  
  TO @out  
  USING Outputters.Tsv(quoting:false, dateTimeFormat:null);
```

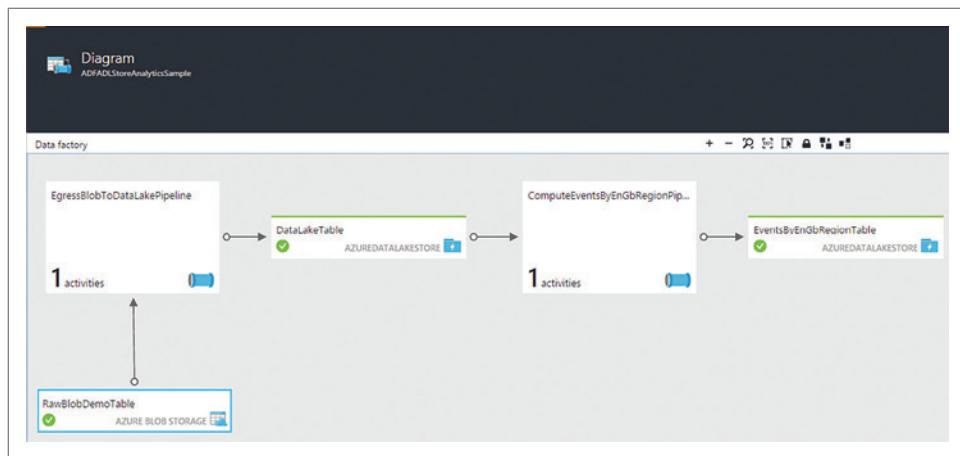


Figure 8 Azure Data Factory Diagram View

Monitoring Big Data Pipelines

The Data Factory service provides a reliable and complete view of storage, processing and data movement services. It helps to quickly assess end-to-end data pipeline health, pinpoint issues and take corrective action, if needed. You can also visually track operational lineage and the relationships between your data across any of your sources, and see a full historical accounting of job execution, system health and dependencies from a single monitoring dashboard. The ADF Diagram view (see **Figure 8**) in the management portal shows the operational lineage of the data factory. You can see two pipelines and the corresponding datasets: EgressBlobToDataLakePipeline (copy data from Azure Blob Storage to Azure Data Lake Store) and ComputeEventsByEnGbRegionPipeline (get all events for the “en-gb” locale and date less than “2012/02/19”).

The ADF copy pipeline in **Figure 8** will start running on 08/08/2015 as the datasets have a daily frequency and the start and

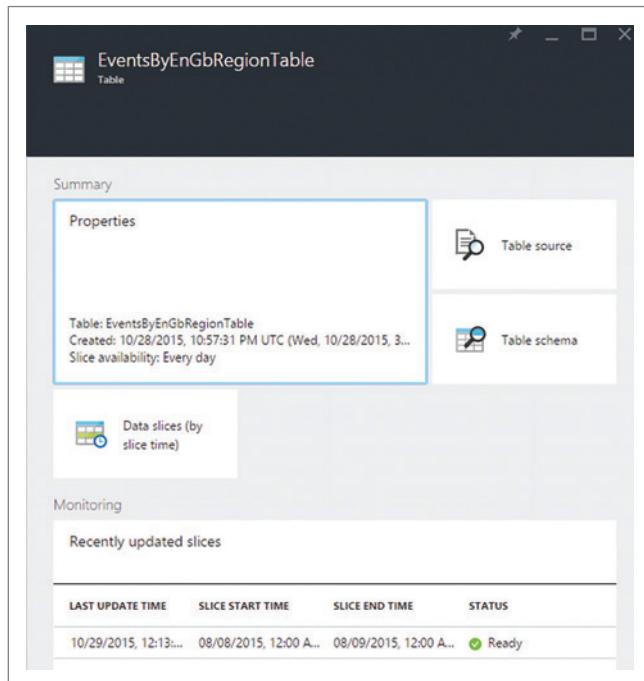


Figure 9 Azure Data Factory Activity View

end parameters in the pipeline definition are both set to 08/08/2015. Therefore, the pipelines will run only for that day and run the U-SQL script just once. You can learn more about scheduling ADF pipelines at bit.ly/1IEVjuM. Click on the EventsByEnGbRegionTable in the Diagram view to see the corresponding activity execution and its status (see **Figure 9**).

You can see that the U-SQL activity in ComputeEventsByEnGbRegionPipeline in ADF has run successfully and created a Result.tsv file (/datalake/output/Result.tsv) in

the AzureDataLakeStore account. The Result.tsv contains all Web log events for the “en-gb” locale and a date less than 2012/02/19. You can also log in to the management portal and use the Azure Data Lake Data Explorer to visualize the Result.tsv file generated (go back to **Figure 4**) as part of the processing step in ADL Store.

You can find detailed documentation about AzureDataLakeAnalyticsU-SQL activity in Azure Data Factory at bit.ly/1WWtxuy.

The Data Factory service provides a reliable and complete view of storage, processing, and data movement services.

Wrapping Up

By following the described steps, you can build an end-to-end Big Data pipeline using Azure Data Factory that allows you to move data to Azure Data Lake Store. You can then use a U-SQL script on the Azure Data Lake Analytics service to do Web log processing. The resulting system can dynamically scale according to your needs, and can be extended to run on a recurring basis. You can also do further downstream processing on the Web log output and move it to another back-end store so the results can be consumed by Power BI or any other BI application your organization uses. Moreover, if you prefer, you can use ADF PowerShell cmdlets, the C# SDK, and the Visual Studio plug-in to build these E2E Big Data pipelines using ADL. Azure Data Lake, together with Azure Data Factory, takes away the complexities normally associated with Big Data in the cloud, ensuring that your current and future business needs can be met. Watch this space for more on solving Big Data problems using Azure services. ■

Gaurav Malhotra is a program manager on the Azure Data Factory team. He lives and works in Redmond, Wash. Reach him at gmal@microsoft.com.

THANKS to the following Microsoft technical experts for reviewing this article:
Omid Afnan, Harish Kumar and Sachin Sheth

One component suite for all Document requirements

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

Using the OneDrive REST API in a UWP App

Laurent Bugnion

In previous frameworks, such as Windows Phone 8, the OneDrive team provided an SDK that was quite comfortable to use, but didn't give the developer a lot of freedom. For example, the login mechanism was only possible using a built-in button control, of which the developer couldn't change the appearance, nor the behavior. But the most difficult part was that this predefined experience didn't let the code be shared between platforms.

Now, however, the OneDrive team provides a modern REST API based on HTTP requests (GET, POST, PUT and so on). This is a flexible way to interact with the gigantic cloud file storage and to build cross-platform code using well-known code-sharing techniques that can run on all Windows platforms, and even on iOS and Android with Xamarin platforms.

This first of a two-part article will show you how to leverage the new OneDrive API to build Universal Windows Platform (UWP)

apps. First, you'll learn how the REST API is working, and how developers are expected to interact with it. You'll see how the user can log into the system using OAuth and how file system operations (browsing folders, getting file information, getting file content, uploading files and so on) can be leveraged. You'll also learn how to execute additional operations, such as accessing the App folder and sharing a link to an item with friends.

In the next article, I'll talk about the OneDrive team's new Portable Class Library (PCL), which is encapsulating the operations described here in a handy library that can be added to your UWP apps, but also in other supported frameworks such as ASP.NET, Xamarin.iOS, Xamarin.Android or Xamarin.Forms.

Note: In addition to the PCL that can be used on the Xamarin.Android and Xamarin.iOS platforms, the OneDrive team also provides native SDKs for the two platforms.

This article discusses:

- Leveraging OneDrive to build universal apps
- How the REST API is working and interacting with it
- Using OAuth and how file system operations can be leveraged
- Executing operations such as accessing the App folder and sharing a link to an item with friends

Technologies discussed:

OneDrive, REST, OAuth, Windows 10, Universal Windows Platform

Code download available at:

galasoft.ch/s/msdnondrive

The Sample Code

You can download sample code from galasoft.ch/s/msdnondrive. It shows how a simple UWP app can use the low-level REST API to perform operations on the OneDrive service. To demonstrate that the code is easily portable, the same app is also implemented for Xamarin.Android, and the same principles can be applied to other platforms.

Understanding the REST API

REST APIs are using HTTP as transport protocol, and rely on the protocol's methods (GET, POST, PUT, DELETE and so on) and on standard error codes (200 Success, 400 Bad request, 500 Server



Document! X

Documentation made easy for:

.NET Assemblies | Web Services (SOAP & REST) | Javascript
SQL/Access/OLE DB Databases | Java | XML Schemas (XSD)
COM Components and Type Libraries

The screenshot shows the Document! X interface integrated with Microsoft Visual Studio. On the left, the Visual Studio Project Explorer displays a .NET assembly named 'System.Messaging'. The code editor shows C# code for the 'RssReader' class, specifically the 'UpdateFeeds' method. The main workspace is the 'Assemblies and Service Layer - Document! X & HelpStudio' window, which contains a detailed documentation view for the 'Message Class' from the 'System.Messaging' namespace. It includes sections like 'Element Type', 'Object Model', and 'Syntax'. A 'Table of Contents' pane is visible on the right. Three callout boxes highlight key features:

- Author Xml format source code comments in a Visual Comment Editor Integrated with Visual Studio.** Points to the integrated XML comment editor in the Visual Studio code editor.
- Author, build and publish documentation in a rich environment including full localization support and Source Control integration for team working and collaboration.** Points to the central documentation editor.
- Generate output in a variety of formats including responsive browser help for web and mobile, CHM and Microsoft Help Viewer for integration with Visual Studio.** Points to the 'Output' tab in the documentation editor.

Trusted by Developers and Technical Writers worldwide since 1998
Download a free trial at <http://www.innovasys.com>

error and so on). Each API entry point is accessible through a unique URI that can also have parameters. In some cases, a simple GET method and a query string can be used to send information to the service and get a result, but it's also possible to build more complex requests by POSTing some objects serialized to JSON.

REST APIs are becoming more and more widespread, and developers should be happy about it as it offers a well-understood way to communicate with Web services. Most important, it makes it possible to build portable components in C# and to use these components on all supported platforms. Another huge advantage is that HTTP is text-based and requests can easily go through firewalls and proxies.

However, communicating with low-level HTTP methods can seem like a lot of work, especially for developers who aren't accustomed with the latest developments in asynchronous programming. Building an asynchronous client using callbacks was not the nicest experience for a developer, because of the deep nesting and possible threading issues that could arise. Thankfully, two relatively recent developments have made this much easier for C# programmers: the HttpClient component and the async/await keywords. For example, accessing the OneDrive Music folder becomes as easy as building a URI and sending a GET request:

```
var uri = new Uri(  
    "https://api.onedrive.com/v1.0/drive/special/music");  
var client = new HttpClient();  
client.DefaultRequestHeaders.Authorization =  
    new AuthenticationHeaderValue("Bearer", AccessToken);  
var json = await client.GetStringAsync(uri);
```

The resulting JSON code can be deserialized (with the JSON.NET library, for example) in order to get a .NET object that's used in the application. This code (admittedly without login or error handling) shows how what would be a complex operation becomes simple and fluid. In addition, this code will run flawlessly on any platform that HttpClient supports, which includes Windows Presentation Foundation (WPF), ASP.NET, Windows 10, Xamarin and more.

Registering Your App

Before a call to the OneDrive API can be made, you must register your app in the OneDrive developer center, configure it and obtain the Client ID key. The registration mechanism is described at bit.ly/1GPBaWL.

When you register your new app, make sure to go to the API Settings page and set the "Mobile or desktop client app" setting to "Yes." All other settings can be left to their default. Then, retrieve the Client ID from the App Settings page, and save it for later.

It's important to understand the different terms and for what each ID is needed:

- **Client ID:** This is a unique ID for your UWP app. You can have multiple client applications connecting to Microsoft services with the same Client ID, but in general it's recommended to use one Client ID per application. The Client ID is linked to information of the UWP app such as its name, the icon and so on. The Client ID is generated when the app is created and never changes.

- **Client Secret:** This is a unique identifier that's generated when the UWP app is created. However, this code may change during the app's lifetime. For example, if you think that the Client Secret was hacked, you can generate a new one, update your apps and the hackers' apps will be denied access. Note

that the Client Secret is generally used in server apps only.

- **User ID and Password:** When the user logs in, he is asked to enter his username and password. Thanks to OAuth, the login interaction happens between the user and OneDrive only, without the client app's knowledge. In practice, this is done using a WebView in which the login dialog is shown.

- **Access Token:** When a user has successfully logged in, the authentication service returns an Access Token. This token is valid for a certain time only (60 minutes), after which the user must log in again. The token must be sent with every request, to prove that the user is authenticated. This authentication mode is named Token Flow in the documentation.

- **Refresh Token:** This token can be requested by the app and saved in order to refresh the Access Token in case it expired. This can be useful if your app is used in background mode for a long time and needs to periodically refresh data without user interaction. This authentication mode is named CodeFlow and is described at bit.ly/1MQ3K0b. In this article, I'll only use Token Flow, which is easier to implement.

Trying Things Out with a Trial Token

If you want to quickly try a few REST requests without implementing authentication first, the OneDrive Dev Center lets you get a trial token, valid for one hour, which can be used by following these steps:

- Go to bit.ly/1MQ3K0b.
- Locate the "Try it now" section and click on the Get Token button.
- If needed, sign in and confirm that you authorize the Dev Center to access your OneDrive account.
- After the login succeeds, a trial token appears in the main Web window.
- Create a new UWP app named TrialOneDrive.
- Open MainPage.xaml and add a button named TrialButton.
- Open MainPage.xaml.cs and add an event handler for the TrialButton Click event as shown in the code in **Figure 1**. Note that this event handler must use the "async" keyword because all OneDrive operations are asynchronous.
- In the code in **Figure 1**, replace the string YOUR TRIAL TOKEN with the trial token that you copied from the Dev Center Web site. Do not copy the words "Authorization: bearer!"
- Place a break point at the last line of the event handler in order to inspect the JSON variable.
- Run the application and click on the button.
- Inspect the retrieved JSON code in a watch window. You should be able to see information about your Music folder, such as its name, creation date, last modified date, child count, the folder's ID (which can be used in various file system operations later), the logged-in username and so on.

Figure 1 Adding an Event Handler for the TrialButton Click Event

```
TrialButton.Click += async (s, e) =>  
{  
    var uri = new Uri(  
        "https://api.onedrive.com/v1.0/drive/special/music");  
    var client = new HttpClient();  
    client.DefaultRequestHeaders.Authorization =  
        new AuthenticationHeaderValue("Bearer", "YOUR TRIAL TOKEN");  
    var json = await client.GetStringAsync(uri);  
}
```

Spreadsheets Made Easy.



Fastest Calculations

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



Comprehensive Charting

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



Windows
Forms



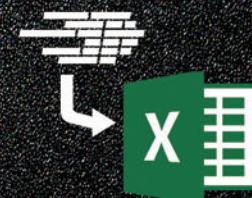
Silverlight



WPF

Powerful Controls

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



Scalable Reporting

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

Download your free fully functional evaluation at SpreadsheetGear.com



SpreadsheetGear

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

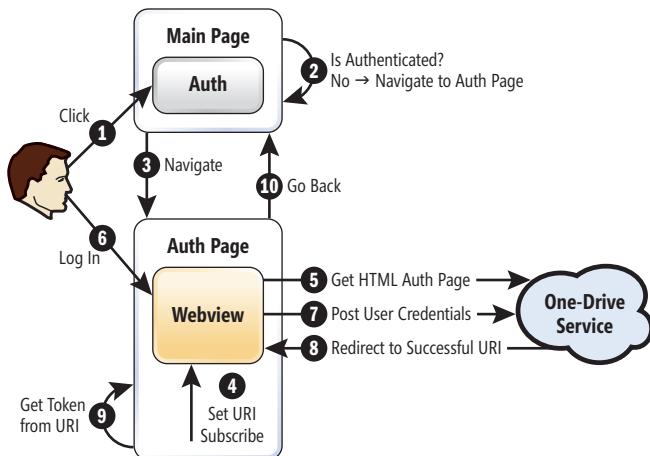


Figure 2 OAuth Workflow

Remember that the trial token expires after an hour, so don't forget to get a new one if you want to demo this application to your boss!

Authenticating with OAuth

Now that you have a registered application and understand the important terms, you can implement authentication. Here are the steps to do that, which are illustrated in **Figure 2**:

1. The user initiates the authentication, for instance by clicking a button.
2. The application checks if an Access Token is available.
3. If yes, the app is already authenticated and the user can move to the next operation. If no Access Token is available, the application navigates to a XAML page containing a WebView.
4. The XAML page sets the WebView's initial URL to load the authentication endpoint's page. The XAML page also subscribes to the WebView's WebNavigationCompleted event, in order to monitor the traffic in this control. You'll learn how the initial URL is structured a little further in the article.
5. The WebView loads the HTML from the authentication endpoint.
6. The user enters his username and password into the HTML authentication page. Note that this interaction is strictly between the user and the OneDrive server. The XAML page acts merely as a container.
7. The user presses the HTML button that submits the form to the authentication server. If the user has set it up, additional pages are shown to handle two-factor authentication.
8. If the credentials are correct, the authentication server redirects the WebView to a special URL that indicates a successful log in. This URL also has the Access Token as one of its query string parameters. The XAML page detects that the redirection occurred, and can parse the Access Token from the WebView's new URL.
9. The Authentication page parses the Access token from the redirected URI.
10. The application navigates back to the original XAML page. The Access Token is saved for further requests.

Note: While it is important to understand how OAuth works, Windows 10 developers are fortunate to have an easier alternative

called `WebAuthenticationBroker`. In this article, I use the "low-level" OAuth mechanism and I'll talk about the `WebAuthenticationBroker` in the next article.

Understanding the Scopes

When the user logs in, the application needs to inform the One-Drive service about the range of features it will need. This is done by specifying scopes in the authentication URL. The service currently supports the following scopes:

- **Single sign-on:** `wl.signin`.
- **Offline access:** `wl.offline_access`. This lets your application get a Refresh Token. This scope is ignored when the Token Flow authentication is used.
- **Readonly access:** `onedrive.readonly`. This gives your application read-only access to the files and folders. If the application attempts to modify a file, or upload a new file, the service returns the error code 403 Forbidden.
- **ReadWrite access:** `onedrive.readwrite`. This gives your application read and write access to the files and folders.
- **AppFolder:** `onedrive.appfolder`. This lets the application access the so-called App Folder. I'll talk more in detail about this special folder later in this article. Note that this scope is automatically granted when you request the readwrite scope. However, requesting the appfolder explicitly will cause this scope to be mentioned in the dialog shown in **Figure 3**, which is a better UX.

After the user enters his credentials in the authentication Web page, he's shown a dialog asking to confirm the permissions requested by the application, as shown in **Figure 4**. The user can change his mind at any time and revoke some of the permissions. To do this, the user logs into his account on the OneDrive Web site, navigates to Security and Privacy and selects the Manage permissions link under Apps & Services. In the sample, you'll always request single sign-on, readwrite and appfolder access.

Creating the Initial URL

The URL used to start the authentication process needs to carry information about the application itself, the features requested (scopes), the authentication mode and the redirect URI.

The authentication mode can be Token or Code. In this article and the sample, I used the Token mode, which means the user

Figure 3 Starting the Authentication

```
if (!_service.CheckAuthenticate()
    async () =>
{
    var dialog = new MessageDialog("You are authenticated!", "Success!");
    await dialog.ShowAsync();
    Frame.GoBack();
},
async () =>
{
    var dialog = new MessageDialog("Problem when authenticating!", "Sorry!");
    await dialog.ShowAsync();
    Frame.GoBack();
})
{
    Frame.Navigate(typeof(AuthenticationPage));
};
```

needs to confirm the login at every new start of the application, or after one hour. This sounds annoying, but in fact the user merely has to confirm the login by pressing the Yes button on the authentication dialog, as shown in **Figure 4**.

The redirect URI is the address to which the OneDrive service will redirect the application when the login is successful. For a Web application, it's the URI of a page in your own application, and is configured in the OneDrive developer center, in the API settings tab. But in the case of a UWP app, you should leave this field empty. Instead, you'll rely on a predefined redirect URI:

http://login.live.com/oauth20_desktop.srf

Putting All the Pieces Together

Now that you understand how the authentication according to OAuth works, let's see some code. You can follow this in the simple sample by clicking on the Authenticate button.

You use a class called OneDriveService, which is implemented in a PCL. The OneDriveService is instantiated in App.xaml.cs, and you pass the Client ID to it.

First, your MainPage asks the OneDrive service if it's already authenticated, which means that an Access Token is already present. You pass two delegates to the CheckAuthenticate method call: an Action which will be called if the authentication is successful, and an Action in case of error.

If the service isn't already authenticated, the MainPage uses its Frame property to navigate to the AuthenticationPage, as shown in **Figure 5**. This page is a simple XAML page with a WebView that takes the whole screen, where the user will enter his username and password and confirm.

When the page is loaded, you get the authentication URI from the OneDriveService:

```
https://login.live.com/oauth20_authorize.srf?client_id=000000004C169646&scope=w1.signin+onedrive.readwrite+onedrive.appfolder&response_type=token&redirect_uri=https%3A%2F%2Flogin.live.com%2Foauth20_desktop.srf
```

As explained, this URI carries all necessary information such as Client ID, Client Secret, Scope, Redirect URI and so on.

Note that the WebView will be redirected a few times before the Access token is returned. This is why you always check the redirection URI in the NavigationCompleted event of the WebView. When the oauth20_desktop.srf URI is finally detected, the OneDriveService will retrieve the Access token from the query string parameters. Additional information (expiration, token type, scope, user ID and so on) is passed in the query string, but you're simply ignoring it here. The Redirect URI with Access token is abbreviated here:

```
https://login.live.com/oauth20_desktop.srf?l1c=1033#access_token=EwB4Aq1D...liiZAE%3d&token_type=bearer&expires_in=3600&scope=w1.signin%20onedrive.readwrite%20onedrive.appfolder%20onedrive.readonly&user_id=8e5323e8b7c7a0928d03e1081531234
```

After the user is authenticated, the application can start interacting with files and folders. In this article, I'll concentrate on a few

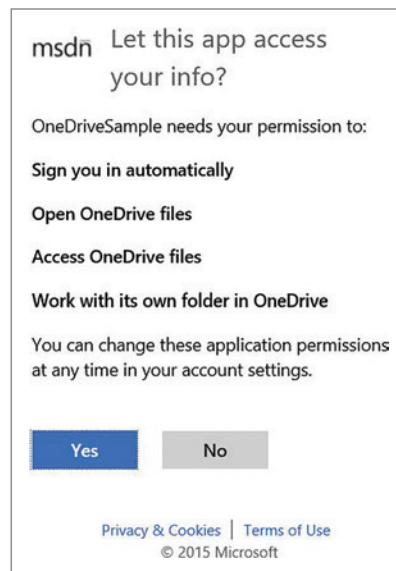


Figure 4 Authentication Confirmation

operations, but extending the application to include additional services is quite easy once the principles are understood.

The Root Folder

First, I'll talk about the Root folder. This folder is unique in your OneDrive, and is the parent to each of your other items.

Per the OneDrive documentation, getting the Root folder can be done using the following request: GET /drive/root. This request will return a JSON response as described at bit.ly/1M5w4NV. The JSON response can be deserialized and the information it contains can be used for further requests.

File and Folder Structure At this point, it's important to understand how the file system is structured. OneDrive relies on a system of facets to provide information about files and folder. For example, a file can also be an

Image and a Photo, and provide additional information about the file (such as width/height for the Image and Camera Model for the Photo). The following list shows the facets that are currently available and some of their most important properties:

- Item (Name, ID, download url, parent reference and so on)
- Folder (child count)
- File
- Audio (album, artist, bit rate, duration, title and so on)
- Image (width, height)
- Photo (camera make and model, taken date time)
- Video (duration, bit rate, width, height)

When you upload a new file to OneDrive, it'll automatically be analyzed and some metadata will be added. For example, a Word

Figure 5 Code for the AuthenticationPage

```
public sealed partial class AuthenticationPage
{
    private readonly OneDriveService _service;

    public AuthenticationPage()
    {
        InitializeComponent();

        _service = ((App)Application.Current).ServiceInstance;

        Loaded += (s, e) =>
        {
            var uri = _service.GetStartUri();
            Web.Navigate(uri);
        };

        Web.NavigationCompleted += (s, e) =>
        {
            if (_service.CheckRedirectUrl(e.Uri.AbsoluteUri))
            {
                _service.ContinueGetTokens(e.Uri);
            }
        };

        Web.NavigationFailed += (s, e) =>
        {
            _service.ContinueGetTokens(null);
        };
    }
}
```

document is just a File. But a picture taken with a camera is a Photo and also an Image, in addition to being a File. Each of these facets carries additional information about the file, as shown in the list.

When you parse the JSON you get from the OneDrive service, you can map these types to C# classes. By inspecting the JSON, you can quite easily find out if an item is a Folder, a File, an Image, a Video and so on. For example, **Figure 6** shows an extract of the JSON response for the Root folder. You can see that the folder property carries information about the folder's child count. If you were getting information about a Photo, you would see the image and photo properties filled up with information about the Width and Height, and about the Camera Make and Model, respectively.

When you upload a new file to OneDrive, it'll automatically be analyzed and some metadata will be added.

In the sample application provided with this article, you deserialize the JSON in the classes contained in the Response folder, which are mapped to the facet system.

Now that you understand how the response works, you can easily access the Root folder information with the code shown here:

```
var client = new HttpClient();
client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", AccessToken);

var uri = new Uri("https://api.onedrive.com/v1.0/drive/root");
var json = await client.GetStringAsync(uri);
var rootFolder = JsonConvert.DeserializeObject<ItemInfoResponse>(json);
```

To see the code in action, you can execute the simple sample and press on the Authenticate button first, and then on the Get Root Folder button.

Getting the Folder's Children Like you see in **Figure 6**, the response only contains meta-information about the folder. You know how many children the folder has, but you need to execute one more request to get the list of children. Here, too, the OneDrive documentation helps and states that you need to request a GET for /drive/items/{item-id}/children, as shown in **Figure 7**.

In **Figure 7**, the JSON is deserialized to a class named ParseChildrenResponse. This is again mapped to the JSON response, which wraps the list of children into a JavaScript array named value. The ParseChildrenResponse class is shown in **Figure 8**.

Because every child is an ItemInfoResponse, you know how to handle these and decide if each child is a folder, a file, a photo, an audio file, etc.

Note that if you want to avoid placing two calls to retrieve the folder information and populate its children list, it's also possible to use the following request, which does everything in one call:

```
GET /drive/items/root?expand=children
```

Understanding Paging When you access a folder with a lot of children, the response sent by the OneDrive service might be incomplete and require paging. By default, the service only returns a list of maximum 200 children. If the folder you're browsing

has more, a property is added to the list of children, called "@odata.nextLink" and can be seen in **Figure 8**, as well. This property contains the URI for the next request that the application needs to do to get to the next "page." In a UWP app, you can choose to either immediately call the service to get the next page of items (which would work OK because long lists are virtualized), or you can show a More button and implement page navigation in the app.

Browsing a Subfolder Once you have a folder's ID, you can easily browse this folder using the following request:

```
GET /drive/items/{item-id}
```

This is great when you already have the folder's ID, but sometimes you don't have this information. Another syntax lets you retrieve the folder's information using its path relative to the root instead. You see this syntax in **Figure 9**.

Figure 6 JSON Response for Root Folder

```
{
  "createdBy": {
    "user": {
      "displayName": "Laurent Bugnion",
      "id": "fb0d8f9700498123"
    }
  },
  "createdDateTime": "2010-11-27T17:09:25.513Z",
  "id": "FB0D8F97004979CD!ABC",
  "lastModifiedBy": {
    "user": {
      "displayName": "Laurent Bugnion",
      "id": "fb0d8f9700498123"
    }
  },
  "lastModifiedDateTime": "2015-10-04T14:36:36.217Z",
  "name": "root",
  "size": 178558187077,
  "webUrl": "https://onedrive.live.com/?cid=fb0d8f9700498123",
  "folder": {
    "childCount": 18
  }
}
```

Figure 7 Getting the List of Children

```
var client = new HttpClient();
client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", AccessToken);

var request = string.Format("/drive/items/{0}/children", info.Id);
var uri = new Uri(
    "https://api.onedrive.com/v1.0/" +
    request);

var json = await client.GetStringAsync(uri);

var response = JsonConvert.DeserializeObject<ParseChildrenResponse>(json);
return response.Value;
```

Figure 8 ParseChildrenResponse Class

```
public class ParseChildrenResponse
{
    public IList<ItemInfoResponse> Value
    {
        get;
        set;
    }

    [JsonProperty("@odata.nextLink")]
    public string NextLink
    {
        get;
        set;
    }
}
```



Whitepaper Available
Four Ways to Optimize ASP.NET Performance

Extreme Performance Linear Scalability

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

In-Memory Distributed Cache

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

ASP.NET Optimization in Web Farms

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



Full Integration with Microsoft Visual Studio

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

Celebrating 10 Years of Market Leadership!



sales@alachisoft.com

US: +1 (925) 236 3830

FREE Download

www.alachisoft.com

The App Folder

OneDrive has a few special folders, such as Music, Documents, Photos and so on. One of the special folders was introduced recently: The App Folder. This is a special folder that's available to your application, for instance to persist roaming settings, upload backups and so on. Note that the App Folder isn't secure or hidden in any way. In fact, it resides in a folder named Apps, located right at the root of the user's OneDrive. The app's App Folder has the same name as the OneDrive application, as entered when you registered it in order to get the Client ID. Note that the application's name might be localized in different languages, based on the

Figure 9 Browsing a Subfolder by Path

```
var client = new HttpClient();
client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", AccessToken);

var request = string.Format("/drive/root:{0}", path);

var uri = new Uri(
    "https://api.onedrive.com/v1.0"
    + request);

var json = await client.GetStringAsync(uri);
var result = JsonConvert.DeserializeObject<ItemInfoResponse>(json);
return result;
```

Figure 10 Downloading a File's Content

```
public async Task<Stream> RefreshAndDownloadContent(
    ItemInfoResponse model,
    bool refreshFirst)
{
    var client = new HttpClient();
    client.DefaultRequestHeaders.Authorization =
        new AuthenticationHeaderValue("Bearer", AccessToken);

    // Refresh the item's information
    if (refreshFirst)
    {
        var request = string.Format("/drive/items/{0}", model.Id);

        var uri = new Uri(
            "https://api.onedrive.com/v1.0"
            + request);

        var json = await client.GetStringAsync(uri);
        var refreshedItem =
            JsonConvert.DeserializeObject<ItemInfoResponse>(json);
        model.DownloadUrl = refreshedItem.DownloadUrl;
    }

    var response = await client.GetAsync(model.DownloadUrl);
    var stream = await response.Content.ReadAsStreamAsync();
    return stream;
}
```

Figure 11 Uploading a File's Content

```
var content = new StreamContent(stream);
var client = new HttpClient();
client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", AccessToken);

var uri = new Uri(
    "https://api.onedrive.com/v1.0"
    + string.Format(
        "/drive/items/{0}:{1}/content",
        parentId,
        fileName));

var response = await client.PutAsync(uri, content);
var json = await response.Content.ReadAsStringAsync();
var result = JsonConvert.DeserializeObject<ItemInfoResponse>(json);
return result;
```

OneDrive application's settings. The user has full access to the App Folder, through the OneDrive Web site or any application and may even delete the folder if he chooses.

Before the concept of App Folder was introduced, applications were (and many still are) saving their settings and other files in the OneDrive root or in a custom folder. Using the App Folder for these items is much cleaner and a better UX. According to the OneDrive documentation (bit.ly/1MBUkS2), the request to get the App Folder's information is:

```
GET /drive/special/approot
```

As you can see, it's easy to implement the App Folder in any of your applications!

Downloading a File

What use would OneDrive be if you couldn't upload or download files? In this section and the next, you'll see how to perform this critical operation, which is extremely easy.

In order to download a file's content, OneDrive creates a DownloadUrl that's saved as a property in the item response. Note, however, that the URL is only valid for a short time, so if the file information was cached, you might need to get the information again from the service first, as shown in **Figure 10**.

Once you get the file content stream, you can process it locally, save it and so on. The simple sample asks the user for a location to save the file using a FileSavePicker.

Uploading a File

Similarly, uploading a file is also quite simple once you get the file's stream. In Windows 10 this can be done with a StorageFile instance, for example with a FileOpenPicker. Once the stream is read, you can upload it to OneDrive. According to the documentation, you need to use a PUT operation:

```
PUT /drive/items/{parent-id}/{filename}/content
```

Figure 12 Getting a Share Link

```
public async Task<LinkResponseInfo> GetLink(
    LinkKind kind,
    string fileId)
{
    // LinkKind id View or Edit

    var client = new HttpClient();
    client.DefaultRequestHeaders.Authorization =
        new AuthenticationHeaderValue("Bearer", AccessToken);

    var request = string.Format("/drive/items/{0}/action.createLink", fileId);
    var uri = new Uri(
        "https://api.onedrive.com/v1.0"
        + request);

    var requestJson = JsonConvert.SerializeObject(
        new RequestLinkInfo
        {
            Type = kind.ToString().ToLower()
        });
    var content = new StringContent(
        requestJson,
        Encoding.UTF8,
        "application/json");

    var response = await client.PostAsync(uri, content);

    var result = JsonConvert.DeserializeObject<LinkResponseInfo>(
        await response.Content.ReadAsStringAsync());
    return result;
}
```

The HttpClient supports the PUT method with an HttpContent instance. In this case, because you have the raw file stream, you can use a StreamContent instance. If all you want to save is a text file, you can use a StringContent and so on.

The other information you need to pass to the service is the ID of the folder in which the file will be saved. In the simple sample shown in **Figure 11**, the parentId is passed to the uploading method. Note that the PUT operation returns a JSON content describing the file's new information on OneDrive, such as its ID, Web URL, download URL and so on.

In addition to the simple upload described here, the OneDrive API also offers multipart and resumable uploads with a different syntax. The simple upload is OK for files smaller than 100MB. If the file is larger, you should consult the documentation at bit.ly/1PB31Cn.

Getting a Unique Link

The last operation that I'll demonstrate here is how to get a unique "share" link for an item. This operation is convenient to send a unique link to a friend over e-mail, SMS, social media and so on. **Figure 12** shows how to get this information, based on the item's ID. In the sample, you can get a link to the file that you just uploaded in the previous demo.

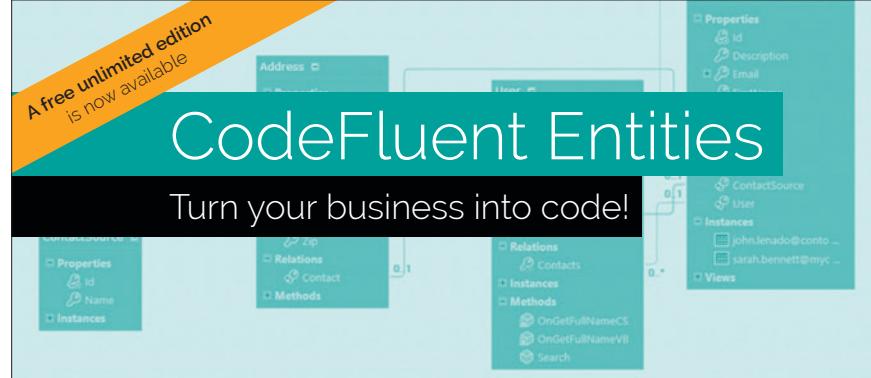
According to the documentation, the request is a little different than the GET requests you placed previously, because the service needs more detailed information. The "share link" request is as follows: POST /drive/items/{item-id}/action.createLink. The information that you need to post to the service is a JSON snippet with the type of the link: "view" or "edit", for example: { "type": "view" }

Wrapping Up

There are a few additional operations that I didn't describe in this article, such as deleting a file, updating a file's content and synchronizing changes. In general, however, the operations all follow the same pattern, and can be performed using HttpClient, HTTP methods and JSON. Thanks to the advancements in asynchronous programming in the Microsoft .NET Framework, such operations are extremely fluid and easy to implement in any supported framework, including Windows 10 but also WPF, Windows Phone, Xamarin.iOS, Xamarin.Android and so on. With such ease of use, there's really no reason to not have some kind of cloud interaction in your UWP apps, for example to back content up or to roam settings. In the next article, you'll see how the freshly released PCL by the OneDrive team will help you make this integration even easier. ■

LAURENT BUGNION is senior director for IdentityMine, one of the leading companies (and a gold partner) for Microsoft technologies. He is based in Zurich, Switzerland. His 2010 book, "Silverlight 4 Unleashed," published by Sams, is an advanced sequel to "Silverlight 2 Unleashed" (2008). He writes for several publications, is in his ninth year as a Microsoft MVP and is in his second year as a Microsoft Regional Director. He's the author of the well-known open source framework MVVM Light for Windows, WPF, Xamarin, and of the popular Pluralsight reference course about MVVM Light. Reach him on his blog at galasoft.ch.

THANKS to the following technical experts for reviewing this article:
Corrado Cavalli (Gaia) and Ryan Gregg (Microsoft)



A free unlimited edition
is now available

CodeFluent Entities

Turn your business into code!

Properties
Relations
Methods

ContactSource
User
Instances
Views

Entities
Relations
Properties
Methods
Contact
Instances
Views

OnGetFullNameCS
OnGetFullNameVB
Search

0..1
0..*

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

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.



Babylon.js: Advanced Features for Enhancing Your First Web Game

Raanan Weber

In the December issue, I began this tutorial by reviewing the basic building blocks of Babylon.js, a WebGL-based 3D game engine (msdn.com/magazine/mt595753). I started designing a very simple bowling game using the tools Babylon.js offers. So far, the game includes the needed objects—a bowling ball, the lane, the gutters and 10 pins.

Now I'll show you how to bring the game to life—how to throw the ball, hit the pins, add some audio effects, provide different camera views and more.

This part of the tutorial naturally relies on and extends the code from the first part. To differentiate between the two parts, I created a new JavaScript file to contain the code that will be used in this part. Even when extending the functionality of a certain object (such as the scene or the main camera), I won't implement it in the function that created the object in the first part. The only function I'll have to extend is init, which contains all of the variables needed for both parts of the tutorial.

This was done for your convenience. When implementing your own game you should, of course, use any programming paradigm

This article discusses:

- Enabling collision detection and gravity
- Using a physics engine for realistic action
- Enhancing the game with audio effects
- Using a dynamic texture to add a score display
- Adding a follow camera for a picture-in-picture effect

Technologies discussed:

Visual Studio 2015 Community Edition, Babylon.js, Oimo.js

Code download available at:

msdn.com/magazine/msdnmag0116

and syntax you wish. I recommend giving TypeScript and its object-oriented style of programming a try. It's wonderful for organizing a project.

In the time it took me to write this article, a new version of Babylon.js was released. I'll continue using version 2.1. To check what's new in Babylon.js 2.2, go to bit.ly/1RC9k6e.

Native Collision Detection

The main camera used in the game is the free camera, which allows the player to travel around the entire 3D scene using mouse and keyboard. Without certain modifications, however, the camera would be able to float through the scene, walk through walls, or even through the floor and the lane. To create a realistic game, the player should be able to move only on the ground and the lane.

To enable this, I'll use Babylon's internal collision detection system. The collision system prevents two objects from merging into each other. It also has a gravity feature that prevents the player from floating in the air if walking forward while looking upward.

First, let's enable collision detection and gravity:

```
function enableCameraCollision(camera, scene) {
    // Enable gravity on the scene. Should be similar to earth's gravity.
    scene.gravity = new BABYLON.Vector3(0, -0.98, 0);
    // Enable collisions globally.
    scene.collisionsEnabled = true;
    // Enable collision detection and gravity on the free camera.
    camera.checkCollisions = true;
    camera.applyGravity = true;
    // Set the player size, the camera's ellipsoid.
    camera.ellipsoid = new BABYLON.Vector3(0.4, 0.8, 0.4);
}
```

This function enables the collision system on the game's scene and camera and sets the camera's ellipsoid, which can be seen as the player's size. This is a box, sizing (in this case) 0.8x1.6x0.8 units, a roughly average human size. The camera needs this box because it's not a mesh. The Babylon.js collision system inspects collisions between meshes only, which is why a mesh should be simulated

for the camera. The ellipsoid defines the center of the object, so 0.4 translates to 0.8 in size. I also enabled the scene's gravity, which will be applied to the camera's movement.

Once the camera has collisions enabled, I need to enable ground and lane collision inspection. This is done with a simple Boolean flag set on each mesh I want to collide against:

```
function enableMeshesCollision(meshes) {
    meshes.forEach(function(mesh) {
        mesh.checkCollisions = true;
    });
}
```

Adding the two function calls to the init function is the final step:

```
// init function from the first part of the tutorial.
...
enableCameraCollision(camera, scene);
enableMeshesCollision([floor, lane, gutters[0], gutters[1]]);
```

The collision detection's only task is to prevent meshes from merging into one another. Its gravity feature was implemented to keep the camera on the ground. To create a realistic physical interaction between specific meshes—in this case the ball and the pins—a more complex system is required: the physics engine.

Throwing the Ball—Physics Integration in Babylon.js

The main action of the game is to throw the ball toward the pins. The requirements are fairly simple: The player should be able to set the direction and strength of the throw. If specific pins are hit, they should fall. If the pins hit other pins, those should fall, as well. If the player throws the ball to the side, it should fall into the gutter. The pins should fall according to the speed with which the ball was thrown at them.

This is exactly the domain of a physics engine. The physics engine calculates the meshes' body dynamics in real time and calculates their next movement according to the applied forces. Simply put, the physics engine is the one who decides what happens to a mesh when another mesh collides with it or when meshes are moved by a user. It takes into account the mesh's current speed, weight, shape and more.

To calculate the rigid body dynamics (the mesh's next movement in space) in real time, the physics engine must simplify the mesh. To do that, each mesh has an impostor—a simple mesh that bounds it (usually a sphere or a box). This reduces the precision of the calculations, but allows a quicker calculation of the physical forces that move the object. To learn more about how the physics engine works, visit bit.ly/1S9A1sU.

The physics engine isn't part of Babylon.js. Instead of committing to a single engine, the framework's developers decided to implement interfaces to different physics engines and let developers decide which one they want to use. Babylon.js currently has interfaces to two physics engines: Cannon.js (cannonjs.org) and Oimo.js (github.com/lo-th/Oimo.js). Both are wonderful! I personally find Oimo integration a bit better and, therefore, will use it in my bowling game. The interface for Cannon.js was completely rewritten for Babylon.js 2.3, which is now in alpha, and now supports the latest Cannon.js version with plenty of bug fixes and new impostors, including complex height maps. I recommend you give it a try if you use Babylon.js 2.3 or later.

Enabling the physics engine is done using a simple line of code:

```
scene.enablePhysics(new BABYLON.Vector3(0, -9.8, 0), new BABYLON.OimoJSPlugin());
```

This sets the gravity of the scene and defines the physics engine to use. Replacing Oimo.js with Cannon.js simply requires changing the second variable to:

```
new BABYLON.CannonJSPlugin()
```

Next, I need to define the impostors on all objects. This is done using the mesh's setPhysicsState function. For example, here is the lane's definition:

```
lane.setPhysicsState(BABYLON.PhysicsEngine.BoxImpostor, {
    mass: 0,
    friction: 0.5,
    restitution: 0
});
```

The first variable is the type of impostor. Because the lane is a perfect box, I use the Box impostor. The second variable is the body's physics definition—its weight (in kilograms), its friction and restitution factor. The lane's mass is 0, as I want it to stay in its location. Setting mass to 0 on an impostor keeps the object locked in its current position.

For the sphere I'll use the Sphere impostor. A bowling ball weighs roughly 6.8 kg and is usually very smooth, so no friction is needed:

```
ball.setPhysicsState(BABYLON.PhysicsEngine.SphereImpostor, {
    mass: 6.8,
    friction: 0,
    restitution: 0
});
```

If you're wondering why I'm using kilograms and not pounds, it's the same reason I'm using meters and not feet throughout the project: The physics engine uses the metric system. For example, the default gravity definition is (0, -9.8, 0), which is approximately Earth's gravity. The units used are meters per second squared (m/s²).

Now I need to be able to throw the ball. For that I'll use a different feature of the physics engine—applying an impulse to a certain object. Here, impulse is a force in a specific direction that's applied to meshes with physics enabled. For example, to throw the ball forward, I'll use the following:

```
ball.applyImpulse(new BABYLON.Vector3(0, 0, 20), ball.getAbsolutePosition());
```

The first variable is the vector of the impulse, here 20 units on the Z axis, which is forward when the scene is reset. The second variable specifies where on the object the force should be applied. In this case, it's the center of the ball. Think about a trick shot in a pool game—the queue can hit the ball at many different points, not only the center. This is how you can simulate such a behavior.

Now I can throw the ball forward. **Figure 1** shows what it looks like when the ball hits the pins.

However, I'm still missing direction and strength.

There are many ways to set the direction. The two best options are to use either the current camera's direction or the pointer's tap position. I'll go with the second option.

Finding the point the user touched in space is done using the PickingInfo object, sent by each pointer-down and pointer-up event. The PickingInfo object contains information about the point the event was triggered on, including the mesh that was touched, the point on the mesh that was touched, the distance to this point and more. If no mesh was touched, the PickingInfo's hit variable will be false. A Babylon.js scene has two useful callback functions that will help me get the picking information: onPointerUp and onPointerDown. Those two callbacks are triggered when pointer events are triggered, and their signature is as follows:

```
function(evt: PointerEvent, pickInfo: PickingInfo) => void
```



Figure 1 The Bowling Ball Hitting the Pins

The evt variable is the original JavaScript event that was triggered. The second variable is the picking information generated by the framework on each event triggered.

I can use those callbacks to throw the ball in that direction:

```
scene.onPointerUp = function(evt, pickInfo) {
    if (pickInfo.hit) {
        // Calculate the direction using the picked point and the ball's position.
        var direction = pickInfo.pickedPoint.subtract(ball.position);
        // To be able to apply scaling correctly, normalization is required.
        direction = direction.normalize();
        // Give it a bit more power (scale the normalized direction).
        var impulse = direction.scale(20);
        // Apply the impulse (and throw the ball).
        ball.applyImpulse(impulse, new BABYLON.Vector3(0, 0, 0));
    }
}
```

Now I can throw the ball in a specific direction. The only thing missing is the strength of the throw. To add that I'll calculate the

Figure 2 Throwing the Ball with Strength and Direction

```
var strengthCounter = 5;

var counterUp = function() {
    strengthCounter += 0.5;
}

// This function will be called on pointer-down events.
scene.onPointerDown = function(evt, pickInfo) {
    // Start increasing the strength counter.
    scene.registerBeforeRender(counterUp);
}

// This function will be called on pointer-up events.
scene.onPointerUp = function(evt, pickInfo) {
    // Stop increasing the strength counter.
    scene.unregisterBeforeRender(counterUp);
    // Calculate throw direction.
    var direction = pickInfo.pickedPoint.subtract(ball.position).normalize();
    // Impulse is multiplied with the strength counter with max value of 25.
    var impulse = direction.scale(Math.min(strengthCounter, 25));
    // Apply the impulse.
    ball.applyImpulse(impulse, ball.getAbsolutePosition());
    // Register a function that will run before each render call
    scene.registerBeforeRender(function ballCheck() {
        if (ball.intersectsMesh(floor, false)) {
            // The ball intersects with the floor, stop checking its position.
            scene.unregisterBeforeRender(ballCheck);
            // Let the ball roll around for 1.5 seconds before resetting it.
            setTimeout(function() {
                var newPosition = scene.activeCameras[0].position.clone();
                newPosition.y /= 2;
                resetBall(ball, newPosition);
            }, 1500);
        }
    });
    strengthCounter = 5;
}
```

delta of the frames between the pointer-down and pointer-up events. **Figure 2** shows the function used to throw the ball with a specific strength.

A note about pointer events—I deliberately haven't used the term "click." Babylon.js uses the Pointer Events system, which extends the mouse-click features of the browser to touch and other input devices capable of clicking, touching and pointing. This way, a touch on a smartphone or a mouse-click on a desktop both trigger the same event. To simulate this on browsers that don't support the feature, Babylon.js uses hand.js, a polyfill for pointer events that's also included in the game's index.html. You can read more about hand.js on its GitHub page at bit.ly/1S4taHF. The Pointer Events draft can be found at bit.ly/1PAdo9J. Note that hand.js will be replaced in future releases with jQuery PEP (bit.ly/1NDMyYa).

That's it for physics! The bowling game just got a whole lot better.

Adding Audio Effects

Adding audio effects to a game provides a huge boost to the UX. Audio can create the right atmosphere and add a bit more realism to the bowling game. Luckily, Babylon.js includes an audio engine, introduced in version 2.0. The audio engine is based on the Web Audio API (bit.ly/1YgBWWQ), which is supported on all major browsers except Internet Explorer. The file formats that can be used depend on the browser itself.

I'll add three different audio effects. The first is the ambient sound—the sound that simulates the surroundings. In the case of a bowling game, the sounds of a bowling hall would normally work nicely, but because I created the bowling lane outside on the grass, some nature sounds will be better.

To add ambient sound, I'll load the sound, auto-play it and loop it constantly:

```
var atmosphere = new BABYLON.Sound("Ambient", "ambient.mp3", scene, null, {
    loop: true,
    autoplay: true
});
```

This sound will play continually from the moment it's loaded.

The second audio effect I'll add is the sound of the ball rolling on the bowling lane. This sound will play as long as the ball is on the lane, but the minute the ball leaves the lane, it will stop.

First, I'll create the rolling sound:

```
var rollingSound = new BABYLON.Sound("rolling", "rolling.mp3", scene, null, {
    loop: true,
    autoplay: false
});
```

The sound will be loaded but won't play until I execute its play function, which will happen when the ball is thrown. I extended the function from **Figure 2** and added the following:

```
...ball.applyImpulse(impulse, new BABYLON.Vector3(0, 0, 0));
// Start the sound.
rollingSound.play();
...
```

I stop the sound when the ball leaves the lane:

```
...
If(ball.intersectsMesh(floor, false)) {
    // Stop the sound.
    rollingSound.stop();
}
...

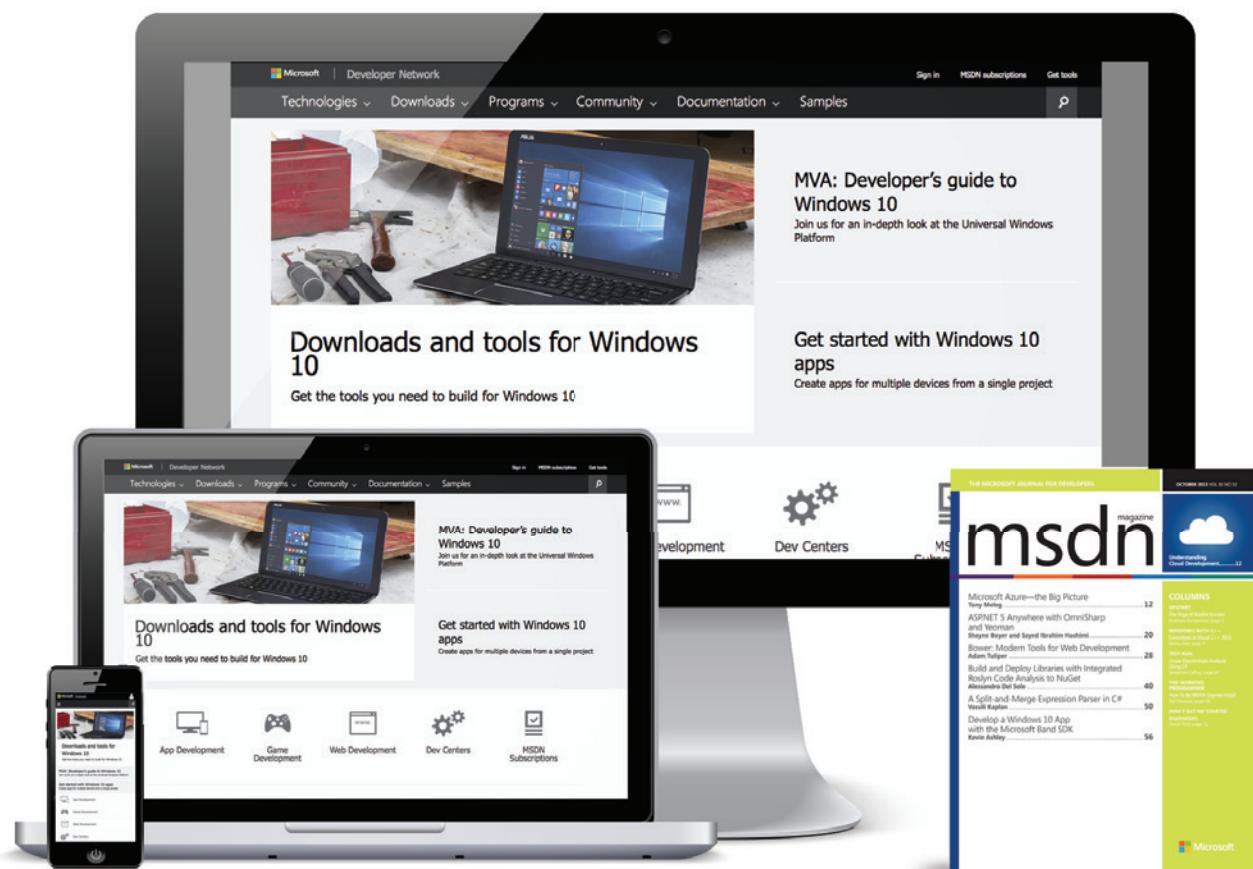
```

Babylon.js lets me attach a sound to a specific mesh. This way it automatically calculates the sound's volume and panning using the

msdn

magazine

Where you need us most.



Visual Studio[®]
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE![®]
360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

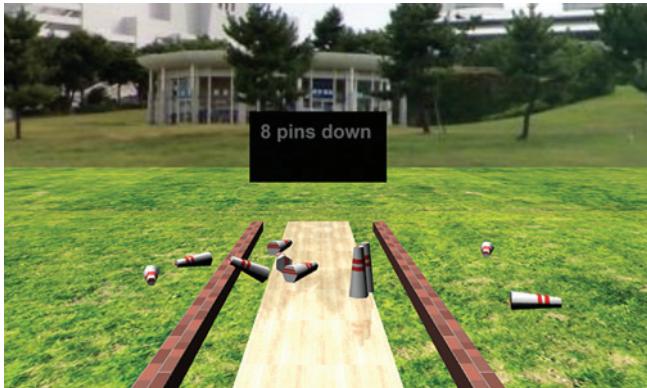


Figure 3 The Game's Scoreboard

mesh's position and creates a more realistic experience. To do this I simply add the following line after I created the sound:

```
rollingSound.attachToMesh(ball);
```

Now the sound will always be played from the ball's position.

The last sound effect I want to add is the ball hitting the pins. To do that I'll create the sound and then attach it to the first pin:

```
var hitSound = new BABYLON.Sound("hit", "hit.mp3", scene);
hitSound.attachToMesh(pins[0]);
```

This sound won't loop and it won't play automatically.

I'll play this sound every time the ball hits one of the pins. To make this happen I'll add a function that, after the ball is thrown, will constantly inspect whether the ball intersects with any pin. If it does intersect, I unregister the function and play the sound. I do so by adding the following lines to the `scene.onPointerUp` function from **Figure 2**:

```
scene.registerBeforeRender(function ballIntersectsPins() {
    // Some will return true if the ball hit any of the pins.
    var intersects = pins.some(function (pin) {
        return ball.intersectsMesh(pin, false);
    });
    if (intersects) {
        // Unregister this function - stop inspecting the intersections.
        scene.unregisterBeforeRender(ballIntersectsPins);
        // Play the hit sound.
        hit.play();
    }
});
```

The game now has all the audio effects I wanted to add. Next, I'll continue to improve the game by adding a scoreboard.

Note that I can't include the audio effects I used with the accompanying project due to the material's copyright. I couldn't find any freely available audio samples that I could also publish. Therefore, the code is commented out. It will work if you add the three audio samples I used.

Figure 4 Resetting the Lane and the Board

```
function clear() {
    // Reset the score.
    score = 0;
    // Initialize the pins.
    initPins(scene, pins);
    // Clear the dynamic texture and draw a welcome string.
    scoreTexture.clear();
    scoreTexture.drawText("welcome!", 120, 100, "bold 72px Arial", "white", "black");
}

scene.actionManager.registerAction(new BABYLON.ExecuteCodeAction({
    trigger: BABYLON.ActionManager.OnKeyUpTrigger,
    parameter: "r"
}, clear));
```

Adding a Score Display

After a player hits the pins, it would be great to actually see how many of them still stand and how many were dropped. For that I'll add a scoreboard.

The scoreboard itself will be a simple black plane with white text on it. To create it, I'll use the dynamic texture feature, which is basically a 2D canvas that can be used as a texture for 3D objects in the game.

Creating the plane and the dynamic texture is simple:

```
var scoreTexture = new BABYLON.DynamicTexture("scoreTexture", 512, scene, true);
var scoreboard = BABYLON.Mesh.CreatePlane("scoreboard", 5, scene);
// Position the scoreboard after the lane.
scoreboard.position.z = 40;
// Create a material for the scoreboard.
scoreboard.material = new BABYLON.StandardMaterial("scoradboardMat", scene);
// Set the diffuse texture to be the dynamic texture.
scoreboard.material.diffuseTexture = scoreTexture;
```

The dynamic texture allows me to directly draw onto the underlying canvas using its `getContext` function, which returns a `CanvasRenderingContext2D` (`mzl.la/1M2mz01`). The dynamic texture object also provides a few help functions that can be useful if I don't want to deal directly with the canvas context. One such function is `drawText`, which lets me draw a string using a specific font on top of this canvas. I will update the canvas whenever the number of dropped pins changes:

```
var score = 0;
scene.registerBeforeRender(function() {
    var newScore = 10 - checkPins(pins, lane);
    if (newScore != score) {
        score = newScore;
        // Clear the canvas.
        scoreTexture.clear();
        // Draw the text using a white font on black background.
        scoreTexture.drawText(score + " pins down", 40, 100,
            "bold 72px Arial", "white", "black");
    }
});
```

Checking if the pins are down is trivial—I check only if their position on the Y-axis is equal to the original Y-position of all pins (the predefined variable '`pinYPosition`')

```
function checkPins(pins) {
    var pinsStanding = 0;
    pins.forEach(function(pin, idx) {
        // Is the pin still standing on top of the lane?
        if (BABYLON.Tools.WithinEpsilon(pinYPosition, pin.position.y, 0.01)) {
            pinsStanding++;
        }
    });
    return pinsStanding;
}
```

The dynamic texture can be seen in **Figure 3**.

All I'm missing now is a function to reset the lane and the board. I'll add an action trigger that will work when the R key is pressed on the keyboard (see **Figure 4**).

Pressing the R key will reset/initialize the scene.

Adding a Follow Camera

A nice effect I want to add to the game is a camera that will follow the bowling ball when it's thrown. I want a camera that will "roll" together with the ball toward the pins and stop when the ball is back in its original position. I can accomplish this using the multi-view feature of Babylon.js.

In the first part of this tutorial I set the free camera object as the scene's active camera using:

```
scene.activeCamera = camera
```

The active camera variable tells the scene which camera is to be rendered, in case there's more than one defined. This is fine when

facebook



Microsoft SharePoint 2010



LinkedIn

SEE THE WORLD AS A DATABASE

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



facebook

Salesforce

Visual Studio

Java

ODBC

Microsoft SQL Server

Excel

BizTalk

MySQL

ODATA

Work With Relational Data, Not Complex APIs or Services

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



Give RSSBus a try today and see what mean:

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

rssbus

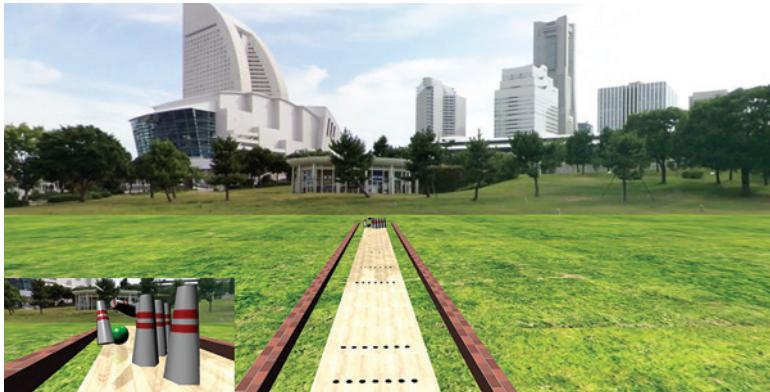


Figure 5 Picture-in-Picture Effect with the Follow Camera

I wish to use a single camera throughout the game. But if I want to have a “picture-in-picture” effect, one active camera isn’t enough. Instead, I need to use the active cameras array stored in the scene under the variable name `scene.activeCameras`. The cameras in this array will be rendered one after the other. If `scene.activeCameras` isn’t empty, `scene.activeCamera` will be ignored.

The first step is to add the original free camera to this array. This is easily done in the init function. Replace `scene.activeCamera = camera;` with:

```
scene.activeCameras.push(camera);
```

In the second step I’ll create a follow camera when the ball is thrown:

```
var followCamera = new BABYLON.FollowCamera("followCamera", ball.position, scene);
followCamera.radius = 1.5; // How far from the object should the camera be.
followCamera.heightOffset = 0.8; // How high above the object should it be.
followCamera.rotationOffset = 180; // The camera's angle. here - from behind.
followCamera.cameraAcceleration = 0.5 // Acceleration of the camera.
followCamera.maxCameraSpeed = 20; // The camera's max speed.
```

This creates the camera and configures it to stand 1.5 units behind and 0.8 units above the object it’s about to follow. This followed object *should* be the ball, but there’s a problem—the ball might spin and the camera will spin with it. What I want to achieve is a “flight path” behind the object. To do that, I’ll create a follow object that will get the ball’s position, but not its rotation:

```
// Create a very small simple mesh.
var followObject = BABYLON.Mesh.CreateBox("followObject", 0.001, scene);
// Set its position to be the same as the ball's position.
followObject.position = ball.position;
```

Then I set the camera’s target to be the `followObject`:

```
followCamera.target = followObject;
```

Now the camera will follow the follow object that’s moving together with the ball.

The last configuration the camera requires is its viewport. Each camera can define the screen real estate it will use. This is done with the `viewport` variable, which is defined using the following variables:

```
var viewport = new BABYLON.Viewport(xPosition, yPosition, width, height);
```

All values are between 0 and 1, just like a percentage (with 1 being 100 percent) relative to the screen’s height and width. The first two values define the camera’s rectangle starting point on the screen, and the width and height define the rectangle’s width and height compared to the screen’s real size. The viewport’s default settings are (0.0, 0.0, 1.0, 1.0), which covers the entire screen. For the follow camera I’ll set the height and width to be 30% of the screen:

```
followCamera.viewport = new BABYLON.Viewport(0.0, 0.0, 0.3, 0.3);
```

Figure 5 shows what the game’s view looks like after a ball is thrown. Notice the view at the bottom-left corner.

The input control—which screen is reacting to input events such as pointer or keyboard events—will stay with the free camera defined in the first part of this tutorial. This was already set in the `init` function.

How to Develop the Game Further

At the start, I said this game would be a prototype. There are a few more things to implement to make it into a real game.

The first would be to add a GUI, one that asks for the user’s name and, for example, shows configuration options (Maybe we can play a bowling game on Mars! Just set the gravity differently.) and whatever else a user might need.

Babylon.js doesn’t offer a native way to create a GUI, but members of the community have created a few extensions you can use to create wonderful GUIs, including CastorGUI (bit.ly/1M2xhD), bGUI (bit.ly/1LCR6jk) and the dialog extension at bit.ly/1MUKpXH. The first uses HTML and CSS to add layers on top of the 3D canvas. The others add 3D dialogs to the scene itself using regular meshes and dynamic textures. I recommend you try these before writing your own solution. They’re all easy to use and will simplify the GUI-creating process a lot.

Another improvement would be to use better meshes. The meshes in my game were all created using Babylon.js internal functions, and clearly there’s a limit to what can be achieved with code only. There are many sites that offer free and paid 3D objects. The best, in my opinion, is TurboSquid (bit.ly/1jSrTzY). Look for low-poly meshes for better performance.

And after adding better meshes, why not add a human object that will actually throw the ball? To do that you’ll need the bones-animation feature integrated in Babylon.js and a mesh that supports the feature. You’ll find a demo showing what it looks like at babylonjs.com/?BONES.

As a final touch, try making the game virtual reality-friendly. The only change needed in this case is the camera used. Replace the `FreeCamera` with a `WebVRFreeCamera` and see how easy it is to target Google Cardboard.

There are many other improvements you could make—adding a camera above the pins, adding more lanes on the side for multi-player functionality, limiting the camera’s movement and the position from which the ball can be thrown and more. I’ll let you discover some of those other features on your own.

Wrapping Up

I hope you had as much fun reading this tutorial as I had writing it, and that it gives you a push in the right direction and gets you to try Babylon.js. This is truly a wonderful framework, built with great love by developers for developers. Visit babylonjs.com for more demos of the framework. And once again, don’t hesitate to join the support forum and ask any question you have. There are no silly questions, just silly answers!

RAANAN WEBER is an IT consultant, full stack developer, husband and father. In his spare time he contributes to Babylon.js and other open source projects. You can read his blog at blog.raananweber.com.

THANKS to the following technical expert for reviewing this article:
David Catuhe

JOIN US on the CAMPAIGN TRAIL in 2016!



MARCH 7 - 11
BALLY'S HOTEL & CASINO
LAS VEGAS, NV

vslive.com/lasvegas

See pages 76-79 for more details



MAY 16 - 19
HYATT AUSTIN, TX

vslive.com/austin

See pages 62-63 for more info



JUNE 13 - 16
HYATT CAMBRIDGE, MA

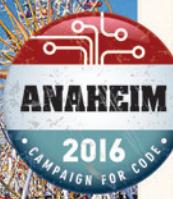
vslive.com/boston

See pages 64-65 for more info



AUGUST 8 - 12
MICROSOFT HQ, REDMOND, WA

vslive.com/redmond



SEPTEMBER 26 - 29
HYATT ORANGE COUNTY, CA -
A DISNEYLAND® GOOD
NEIGHBOR HOTEL

vslive.com/anaheim



OCTOBER 3 - 6
RENAISSANCE, WASHINGTON, D.C.

vslive.com/dc



PART OF LIVE! 360

DECEMBER 5 - 9
LOEWS ROYAL PACIFIC
ORLANDO, FL



CONNECT WITH VISUAL STUDIO LIVE!

[@VSLive](https://twitter.com/vslive)



[facebook.com – Search "VSLive"](https://facebook.com/Search%20%22VSLive%22)



[linkedin.com – Join the "Visual Studio Live" group!](https://linkedin.com – Join the %22Visual%20Studio%20Live%22%20group)

TURN THE PAGE FOR
MORE EVENT DETAILS.

CAMPAIGN FOR CODE 2016 • VISUAL STUDIO LIVE!

Austin

COUNTS



Visual Studio[®]
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE!



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.

SUPPORTED BY



PRODUCED BY



AUSTIN • MAY 16-19, 2016

HYATT REGENCY • AUSTIN, TX



CONNECT WITH VISUAL STUDIO LIVE!



[@VSLive](https://twitter.com/vslive)



[facebook.com – Search "VSLive"](https://facebook.com/Search%20%22VSLive%22)



[linkedin.com – Join the "Visual Studio Live" group!](https://linkedin.com – Join the %22Visual Studio Live%22 group!)

TOPICS INCLUDE:

- Visual Studio / .NET
- Windows Client
- Mobile Client
- JavaScript/HTML5 Client
- ASP.NET
- Cloud Computing
- Database & Analytics
- ALM / DevOps
- UX/Design

Get out and code. Register to join us today!

**REGISTER NOW
AND SAVE \$300!**



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

USE PROMO CODE VSLJAN5

VSLIVE.COM/AUSTIN

BETTER

CAMPAIGN FOR CODE 2016 • VISUAL STUDIO LIVE!

CODE

Boston

FOR ALL



Visual Studio
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE!



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.

SUPPORTED BY



PRODUCED BY



BOSTON • JUNE 13-16, 2016

HYATT REGENCY • CAMBRIDGE, MA



CONNECT WITH VISUAL STUDIO LIVE!



twitter.com/vslive – @VSLive



facebook.com – Search "VSLive"



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

TOPICS INCLUDE:

- Visual Studio / .NET
- Windows Client
- Mobile Client
- JavaScript/HTML5 Client
- ASP.NET
- Cloud Computing
- Database & Analytics
- ALM / DevOps
- UX/Design

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

**REGISTER NOW
AND SAVE \$300!**



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

USE PROMO CODE VSLJANS

VSLIVE.COM/BOSTON



How To Be MEAN: Test Me MEANly

Howdy, "Nodeists." Welcome back to the continuing saga of input, output, databases, users, UIs, APIs and parallel universes. (See the first part in this series [msdn.com/magazine/mt185576] if you missed that reference.)

In the previous installment (msdn.com/magazine/mt595757), the application had grown to the point where the API endpoint being built (a super-simplified "persons" database) rounded out its CRUD facilities to include the CUD, where the previous installment had provided just the "R." Building out in small steps is important, certainly, but it's nice to have something that can see changes to it and reflect them back when asked about them.

This brought up an uncomfortable point. When building a Web API such as this, "easy testing," in which a developer can just fire up the Web page and eyeball it to determine whether something is working correctly, isn't an option here. Well, sure, as pointed out last time, using tools such as cURL, Postman or Rumscope can help make it easier to eyeball it, but frankly, trusting eyeballs isn't really the point. The application needs some automated tests to help make sure everything is running smoothly and correctly. It should be a test suite that's easy to run, covers the code well and has all the other characteristics that so many others have documented far better than I can.

One note before I go too deep down this path, though. There's a bit of debate around "unit" tests, which have intricate knowledge of the inner workings of the code and (usually) require some level of mocking (in this case, the Express request and response objects, for example). These are supposed to be small, focused and execute quickly so they can be used as part of a build process (right after compilation, for compiled languages like C#) without slowing down programmers and jerking them out of the flow of their thoughts. Another view is that of "integration" or "feature" tests, which are more "external" tests, meaning the test relies on the frameworks to do their thing and treats the layer as a more opaque entity. For Web APIs, I tend to favor the latter; I've found that trying to write unit tests over Express controllers can often be more work than benefit to get all the mocking right. Given that controllers are generally intended to be single-purpose, anyway, and there are few, if any, bugs I've found in the Express layer that interferes with testing my own code, I've found external tests to be a better value in the bang-for-your-buck discussion.

With that out of the way, it's time to write some tests that exercise the persons API I've built so far. And to that end, there are two Node.js packages that immediately come to mind: mocha and supertest.

Getting Tested

Of course, as with all things Nodeish, life begins with npm; in this case, both mocha and supertest will need to be added to the project via npm install:

```
npm install --save-dev mocha supertest
```

(While it's traditional to do each separately, there's no reason not to do them together like that, particularly in this case where I already know I want to use both.)

Before I pass beyond that, note the --save-dev argument: this puts the two into the devDependencies section of the package.json file, meaning they're development-only dependencies and, as a result, when cloud hosts (such as Microsoft Azure, for example) prepare the code for execution in production, they'll leave those out.

The Node.js community treats its testing frameworks much as it does the rest of its projects: as small bits, looking to be composed together in developer-defined ways, to create a whole that is, presumably, pleasing to the developer. In this case, testing starts with the mocha package, which is a testing package designed to make "asynchronous testing simple and fun," according to its Web site. It will act as the basic "scaffolding" if you will, of the testing effort. But because the things under test will be Web APIs (and because I don't want to be writing all this low-level HTTP code), the second library, supertest, will add some useful HTTP-specific testing behavior. There's one more optional part to the set, the should library, which I choose to add because I prefer Behavior-Driven Development (BDD)-style should assertions. But the assert package, which comes with mocha, is also there for those who prefer traditional assert styles.

Once both packages are installed, testing begins, not surprisingly, by writing the first test. Although not required, it's common for

Figure 1 Setting Up Some Baseline Tests Just for Fun

```
var supertest = require('supertest');
var express = require('express');
var assert = require('assert');
var should = require('should');

describe('Baseline tests', function() {
  describe('#indexOf()', function () {
    it('should return -1 when the value is not present', function () {
      assert.equal(-1, [1,2,3].indexOf(5));
      assert.equal(-1, [1,2,3].indexOf(0));
    });
  });
});
```

WASHINGTON, DC

JUNE
8-9 | 2016

Tracks include:



ACQUIRE

Acquisition & Management Show

ACQUIRE is a new 2-day event that focuses on 3 key OMB spending categories—**Professional Services**, **Office Management** and **Information Technology**. Covering all aspects of the acquisition and management process, from setting policy and defining requirements to implementing and managing programs to end user experience, it's guaranteed to be **THE NEXT BIG THING**.

Start planning your presence now.
Exhibit & Sponsorship packages available.

Contact Stacy Money for pricing & details:
smoney@1105media.com | 415.444.6933

ACQUIREshow.com

Node.js projects to create a separate subdirectory in which the test code lives, called, not surprisingly, test. Into this directory goes the first test file, getTest.js, as shown in **Figure 1**.

There are a couple of things that require some explanation before I move on. The describe methods are simple output/execution pairs—the string is the message that should be displayed in the testing console (typically the Command Prompt or Terminal window) and the function is what should be run as the test. Notice that describe blocks can be nested—this can be helpful to split up tests along logical lines, particularly because you can then use the skip function to turn off whole blocks of them should that be necessary. The it function then describes a single test—in this case, to test that the JavaScript array's indexOf method functions as it's supposed to, and again, the string passed into "it" is the description of the test printed to the test console.

The nested function inside the it call is, not surprisingly, the actual test, and here I've chosen to show both the traditional assert style of assertions, as well as the more BDD-like should-style assertions, which is a more fluent API. Frankly, there are other options (the mocha Web page lists two more besides these two), but because I like should, that's what I'm going to use. However, the should package will need to be installed via npm, whereas assert comes along for the ride already with mocha and supertest.

For a quick test, just type "mocha" into the directory containing app.js and it will pick up the tests in the test subdirectory automatically. But Node.js has a slightly better convention, which is to populate the scripts collection in the package.json file to have a set of command-line parameters to npm for easy execution; "npm start" can then start the server, and "npm test" can run the tests without the administrator (or cloud system) having to know what was used to build the code. This just requires filling out the "scripts" collection accordingly inside of package.json:

```
{  
  "name": "MSDN-MEAN",  
  "version": "0.0.1",  
  "description": "Testing out building a MEAN app from scratch",  
  "main": "app.js",  
  "scripts": {  
    "start": "node app.js",  
    "test": "mocha"  
  },  
  // ...  
}
```

So now, a quick "npm test" will launch the tests and find out that, yes, indexOf works correctly.

Figure 2 Get Me All the Persons!

```
var request = supertest("http://localhost:3000");  
describe('/person tests', function() {  
  
  it('should return Ted for id 1', function(done) {  
    request  
      .get('/persons/1')  
      .expect(200)  
      .expect('Content-Type', /json/)  
      .expect(function(res) {  
        res.body.id.should.equal(1)  
        res.body.firstName.should.equal("Ted")  
        res.body.lastName.should.equal("Neward")  
        res.body.status.should.equal("MEANING")  
      })  
      .end(done);  
  });  
});
```

Having done that, let's turn those tests off. (Obviously, you could remove them entirely, but I like having a baseline that I know works, in case something fails with my setup.) That's easily done by using skip as part of the chained describe, as shown here:

```
describe.skip('Baseline tests', function() {  
  describe('#indexOf()', function () {  
    it('should return -1 when the value is not present', function () {  
      assert.equal(-1, [1,2,3].indexOf(5));  
      assert.equal(-1, [1,2,3].indexOf(0));  
  
      [1,2,3].indexOf(5).should.equal(-1);  
      [1,2,3].indexOf(0).should.equal(-1);  
    });  
  });  
});
```

This will now skip these tests—they're still listed to the test console, but there's no checkmark next to them to indicate the tests were run.

Testing APIs

That was fun, but indexOf was already pretty well tested; the GET /persons endpoint isn't ... yet.

Once both packages are installed, testing begins, not surprisingly, by writing the first test.

This is where supertest comes into play; using it, you can create a tiny HTTP agent that knows how to query the endpoint (in this case, using the server settings of "localhost" and port 3000, as established by the app code written before this) and verify its results. This means there are a couple of things that need to be added; first, the app code has to be loaded and run as part of the test and that's done by requiring the app.js file in the parent directory from the first line of the test:

```
require("../app.js")
```

This will get the app started (which is easy to spot if the DEBUG environment variable is set to app, remember) and running.

Then, using the supertest library to create a request object against the local host server and port 3000, you can use that to issue a request against the /persons endpoint with a person ID of 1 (making it /persons/1, remember) and verify that it comes back OK (a 200 status code), comes back as JSON and the body of the JSON contains what's expected, as shown in **Figure 2**.

Notice how the request object has a fluent API of its own, using a verb method style similar to that of Express, so get('/persons/1') translates into a GET request against that URL. The successive expect methods, however, do exactly what it sounds like: They expect a particular kind of result, and if that isn't the case, it fails the entire test. However, the last end method is important to note because it makes use of the one parameter passed into "it": the "done" object, which is an opaque callback that signals that the test is finished. Because supertest does all of this testing serially but asynchronously (in the Node.js fashion), there needs to be some signal that the test is completed—if done isn't used, then after two seconds (per each test), supertest will assume the test timed out and signal an error.

Last, because space is getting short for this one, let's verify that you can add persons to the database by using the post method, and

Figure 3 Verifying That Adding a Person Actually Works

```
it('should allow me to add a person', function(done) {
  request
    .post('/persons')
    .send({'firstName':'Ted',
           'lastName':'Pattison','status':'SharePointing'})
    .expect(200)
    .expect('Content-Type',/json/)
    .expect(function(res) {
      should.exist(res.body.id);
      res.body.firstName.should.equal("Ted");
      res.body.lastName.should.equal("Pattison");
    })
    .end(done);
})
```

sending along a new JSON object containing the person instance that needs to be added to the system, as shown in **Figure 3**.

Easy, right? Note that the first line of the body-verification expect is to test that the id field exists. However, the should library can't trust that it won't be undefined, and calling a method or property on an undefined generates an exception, so you need to use should directly to verify it exists. Aside from that, though, it reads nicely.

The two other methods that require testing, PUT and DELETE, are pretty easy to write. The PUT endpoint test will use put instead of post, and the URL will need to point to a singular person URL (/persons/1 for the person "Ted" "Neward," for example). The JSON body to send would look the same as the POST case, with differing values (because the test is to see if those updates will be accepted), and then test that the returned JSON body reflects those updates (because the test is also to see if those updates will be reflected in the new object returned). And the DELETE endpoint test will use the delete method, passing in the URL for a specific person (/persons/1), and this time, pass in neither a JSON body nor expect one returned. (Although it seems a common API convention to return the deleted object from a DELETE endpoint, to be honest, I personally have never really seen much point to it, and have never used the returned JSON body for any reason.)

With those five tests in place, and armed with the newly enhanced npm script tag in package.json, it remains only to run npm test, watch the rest runner do its thing, see no errors, do a source code commit-and-push, and update the Azure environment with the new code.

Wrapping Up

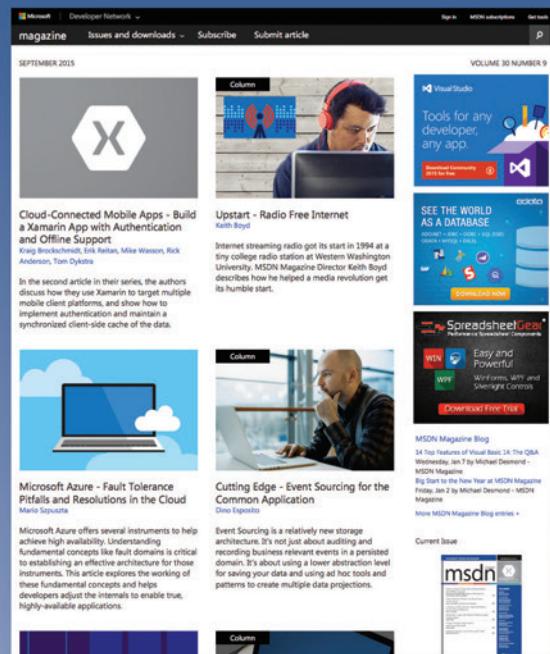
There's a bunch of material about mocha, supertest and should that I don't have space to get into—for example, like most testing frameworks, mocha supports before and after hooks that will run around each test; also, the supertest library supports cookie-based requests, which is useful when working with HTTP endpoints that use cookies to maintain some kind of session state. Fortunately, the documentation around each of those libraries is pretty good, and examples floating around the Web are legion. Alas, for now, it's time to step back and say ... happy coding! ■

TED NEWARD is the principal at Neward & Associates, a Seattle-based polytechnology consulting firm. He has written more than 100 articles, is an F# MVP and has a dozen books to his name. 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

msdnmagazine.com

MSDN Magazine Online



It's like *MSDN Magazine*—only better. In addition to all the great articles from the print edition, you get:

- Code Downloads
- The *MSDN Magazine* Blog
- Digital Magazine Downloads
- Searchable Content

All of this and more at
msdn.microsoft.com/magazine

msdn
magazine



C# Scripting

With the arrival of Visual Studio 2015 Update 1, henceforth Update 1, comes a new C# read-evaluate-print-loop (REPL), available as a new interactive window within Visual Studio 2015 or as a new command-line interface (CLI) called CSI. In addition to bringing the C# language to the command line, Update 1 also introduces a new C# scripting language, traditionally saved into a CSX file.

Before delving into the details of the new C# scripting, it's important to understand the target scenarios. C# scripting is a tool for testing out your C# and .NET snippets without the effort of creating multiple unit testing or console projects. It provides a lightweight option for quickly coding up a LINQ aggregate method call on the command line, checking the .NET API for unzipping files, or invoking a REST API to figure out what it returns or how it works. It provides an easy means to explore and understand an API without the overhead of a yet another CSPROJ file in your %TEMP% directory.

The C# REPL Command-Line Interface (CSI.EXE)

As with learning C# itself, the best way to get started with learning the C# REPL interface is to run it and begin executing commands. To launch it, run the command csi.exe from the Visual Studio 2015 developer command prompt or use the full path, C:\Program Files (x86)\MSBuild\14.0\bin\csi.exe. From there, begin executing C# statements such as those shown in **Figure 1**.

C# scripting is a tool for testing out your C# and .NET snippets without the effort of creating multiple unit testing or console projects.

The first thing to note is the obvious—it's like C#—albeit a new dialect of C# (but without the ceremony that's appreciated in a full production program and unnecessary in a quick-and-dirty prototype). Therefore, as you'd expect, if you want to call a static method you can write out the fully qualified method name and pass arguments within parentheses. As in C#, you declare a variable by prefixing it with the type and, optionally, assigning it a new value at declaration time. Again, as you'd expect, any valid method

Figure 1 CSI REPL Sample

```
C:\Program Files (x86)\Microsoft Visual Studio 14.0>csi
Microsoft (R) Visual C# Interactive Compiler version 1.1.0.51014
Copyright (C) Microsoft Corporation. All rights reserved.

Type "#help" for more information.
> System.Console.WriteLine("Hello! My name is Inigo Montoya");
Hello! My name is Inigo Montoya
>
> ConsoleColor originalConsoleColor = Console.ForegroundColor;
> try{
.   Console.ForegroundColor = ConsoleColor.Red;
.   Console.WriteLine("You killed my father. Prepare to die.");
. }
. finally
. {
.   Console.ForegroundColor = originalConsoleColor;
. }
You killed my father. Prepare to die.
> IEnumerable<Process> processes = Process.GetProcesses();
> using System.Collections.Generic;
> processes.Where(process => process.ProcessName.StartsWith("c") );
. Select(process => process.ProcessName ).Distinct()
DistinctIterator { "chrome", "csi", "cmd", "conhost", "csrss" }
> processes.First(process => process.ProcessName == "csi" ).MainModule.
FileName
"C:\Program Files (x86)\MSBuild\14.0\bin\csi.exe"
> $"The current directory is { Environment.CurrentDirectory }."
"The current directory is C:\Program Files (x86)\Microsoft Visual
Studio 14.0."
>
```

body syntax—try/catch/finally blocks, variable declaration, lambda expressions and LINQ—works seamlessly.

And even on the command line, other C# features, such as string constructs (case sensitivity, string literals and string interpolation), are maintained. Therefore, when you're using or outputting paths, backslashes need to be escaped using a C# escape character ("\") or a string literal, as do double backslashes in the output of a path like that of csi.exe. String interpolation works, too, as the "current directory" sample line in **Figure 1** demonstrates.

C# scripting allows far more than statements and expressions, though. You can declare custom types, embed type metadata via attributes, and even simplify verbosity using C# script-specific declaratives. Consider the spell-checking sample in **Figure 2**.

For the most part, this is just a standard C# class declaration. However, there are several specific C# scripting features. First, the #r directive serves to reference an external assembly. In this case, the reference is to Newtonsoft.Json.dll, which helps parse the JSON data. Note, however, this is a directive designed for referencing files in the file system. As such, it doesn't require the unnecessary ceremony of a backslash escape sequence.

Second, you can take the entire listing and save it as a CSX file, then “import” or “inline” the file into the C# REPL window using #load Spell.csx. The #load directive allows you to include additional script files as if all the #load files were included in the same “project” or “compilation.” Placing code into a separate C# script file enables a type of file refactoring and, more important, the ability to persist the C# script over time.

Using declarations are another C# language feature allowed in C# scripting, one that **Figure 2** leverages several times. Note that just as in C#, a using declaration is scoped to the file. Therefore, if you called #load Spell.csx from the REPL window, it wouldn’t persist the using Newtonsoft.Json declarative outside the Spell.csx. In other words, using Newtonsoft.Json from within Spell.csx wouldn’t persist into the REPL window without being declared again explicitly in the REPL window (and vice versa). Note that the C# 6.0 using static declarative is also supported. Therefore, a “using static System.Console” declarative eliminates the need to prefix any of the System.Console members with the type, allowing for REPL commands such as “WriteLine(“Hello! My name is Inigo Montoya”).”

Other constructs of note in C# scripting include the use of attributes, using statements, property and function declaration, and support for async/await. Given the latter support, it’s even possible to leverage await in the REPL window:

```
(await Spell.CheckAsync("entrepreneur")).IsCorrect
```

Here are some further notes about the C# REPL interface:

- You can’t run csi.exe from within Windows PowerShell Integrated Scripting Environment (ISE) as it requires direct

Figure 2 The C# Scripting Class Spell (Spell.csx)

```
#r ".\Newtonsoft.Json.7.0.1\lib\net45\Newtonsoft.Json.dll"
#load "Mashape.csx" // Sets a value for the string Mashape.Key

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Threading.Tasks;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

public class Spell
{
    [JsonProperty("original")]
    public string Original { get; set; }
    [JsonProperty("suggestion")]
    public string Suggestion { get; set; }

    [JsonProperty(PropertyName = "corrections")]
    private JObject InternalCorrections { get; set; }

    public IEnumerable<string> Corrections
    {
        get
        {
            if (!IsCorrect)
            {
                return InternalCorrections?[Original].Select(
                    x => x.ToString()) ?? Enumerable.Empty<string>();
            }
            else return Enumerable.Empty<string>();
        }
    }
    public bool IsCorrect
    {
        get { return Original == Suggestion; }
    }
}
```

console input, which isn’t supported from the “simulated” console windows of Windows PowerShell ISE. (For this reason, consider adding to the unsupported list of console applications—\$psUnsupportedConsoleApplications.)

- There is no “exit” or “quit” command to leave the CSI program. Instead, you use Ctrl+C to end the program.

Similarly, compilation occurs in real time, as you edit, so syntax errors and the like will automatically be red-squiggle-underlined.

- Command history is persisted between csi.exe sessions launched from the same cmd.exe or PowerShell.exe session. For example, if you start csi.exe, invoke Console.WriteLine("HelloWorld"), use Ctrl+C to exit, and then relaunch csi.exe, the up arrow will display the previous Console.WriteLine("HelloWorld") command. Exiting the cmd.exe window and relaunching it will clear out the history.
- Csi.exe supports the #help REPL command, which displays the output shown in **Figure 3**.
- Csi.exe supports a number of command-line options, as shown in **Figure 4**.

```
static public bool Check(string word, out IEnumerable<string> corrections)
{
    Task<Spell> taskCorrections = CheckAsync(word);
    corrections = taskCorrections.Result.Corrections;
    return taskCorrections.Result.IsCorrect;
}

static public async Task<Spell> CheckAsync(string word)
{
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(
        $"https://montanaflynn-spellcheck.p.mashape.com/check/?text={ word }");
    request.Method = "POST";
    request.ContentType = "application/json";
    request.Headers = new WebHeaderCollection();
    // Mashape.Key is the string key available for
    // Mashape for the montanaflynn API.
    request.Headers.Add("X-Mashape-Key", Mashape.Key);

    using (HttpWebResponse response =
        await request.GetResponseAsync() as HttpWebResponse)
    {
        if (response.StatusCode != HttpStatusCode.OK)
            throw new Exception(String.Format(
                "Server error (HTTP {0}: {1}).",
                response.StatusCode,
                response.StatusDescription));
        using(Stream stream = response.GetResponseStream())
        using(StreamReader streamReader = new StreamReader(stream))
        {
            string strsb = await streamReader.ReadToEndAsync();
            Spell spell = Newtonsoft.Json.JsonConvert.DeserializeObject<Spell>(strsb);
            // Assume spelling was only requested on first word.
            return spell;
        }
    }
}
```

Figure 3 The REPL #help Command Output

```
> #help
Keyboard shortcuts:
Enter      If the current submission appears to be complete, evaluate it.
            Otherwise, insert a new line.
Escape      Clear the current submission.
UpArrow    Replace the current submission with a previous submission.
DownArrow   Replace the current submission with a subsequent
            submission (after having previously navigated backward).

REPL commands:
#help      Display help on available commands and key bindings.
```

As noted, csi.exe allows you to specify a default “profile” file that customizes your command window:

- To clear the CSI console, invoke Console.Clear. (Consider a using static System.Console declarative to add support for simply invoking Clear.)
- If you’re entering a multi-line command and make an error on an earlier line, you can use Ctrl+Z followed by Enter to cancel and return to an empty command prompt without executing (note that the ^Z will appear in the console).

The Visual Studio C# Interactive Window

As I mentioned, there’s also a new Visual Studio C# Interactive window in Update 1, as shown in **Figure 5**. The C# Interactive window is launched from the View | Other Windows | C# Interactive menu, which opens up an additional docked window. Like the csi.exe window, it’s a C# REPL window but with a few added features. First, it includes syntax color coding and IntelliSense. Similarly, compilation occurs in real time, as you edit, so syntax errors and the like will automatically be red-squiggle-underlined.

A common association with the C# Interactive Window is, of course, the Visual Studio Immediate and Command windows. While there are overlaps—after all, they’re both REPL windows against which you can execute .NET statements—they have significantly different purposes. The C# Immediate Window is bound directly to the debug context of your application, thus allowing you to inject additional statements into the context, examine data within the debug session, and even manipulate and update the data and debug context. Similarly, the Command Window provides a CLI

Figure 4 Csi.exe Command-Line Options

```
Microsoft (R) Visual C# Interactive Compiler version 1.1.0.51014
Copyright (C) Microsoft Corporation. All rights reserved.

Usage: csi [option] ... [script-file.csx] [script-argument] ...

Executes script-file.csx if specified, otherwise launches an interactive
REPL (Read Eval Print Loop).

Options:
/help      Display this usage message (alternative form: /?)
/i         Drop to REPL after executing the specified script
/r:<file>  Reference metadata from the specified assembly file
            (alternative form: /reference)
/r:<file list> Reference metadata from the specified assembly files
            (alternative form: /reference)
/lib:<path list> List of directories where to look for libraries specified
            by #r directive (alternative forms: /libPath /libPaths)
/u:<namespace> Define global namespace using
            (alternative forms: /using, /usings, /import, /imports)
@<file>    Read response file for more options
--         Indicates that the remaining arguments should not be
            treated as options
```

for manipulating Visual Studio, including executing the various menus, but from the Command Window rather than from the menus themselves. (Executing the command View.C#Interactive, for example, opens the C# Interactive Window.) In contrast, the C# Interactive Window allows you to execute C#, including all the features relating to the C# REPL interface discussed in the previous section. However, the C# Interactive Window doesn’t have access to the debug context. It’s an entirely independent C# session without handles to either the debug context or even to Visual Studio. Like csi.exe, it’s an environment that lets you experiment with quick C# and .NET snippets to verify your understanding without having to start yet another Visual Studio console or unit testing project. Instead of having to launch a separate program, however, the C# Interactive Window is hosted within Visual Studio, where the developer is presumably already residing.

The C# Interactive window includes syntax color coding and IntelliSense.

Here are some notes about the C# Interactive Window:

- The C# Interactive Window supports a number of additional REPL commands not found in csi.exe, including:
 - #cls/#clear to clear the contents of the editor window
 - #reset to restore the execution environment to its initial state while maintaining the command history
- The keyboard shortcuts are a little unexpected, as the #help output in **Figure 6** shows.

It’s important to note that Alt+UpArrow/DownArrow are the keyboard shortcuts for recalling the command history. Microsoft selected these over the simpler UpArrow/DownArrow because it wanted the Interactive window experience to match that of a standard Visual Studio code window.

- Because the C# Interactive Window is hosted within Visual Studio, there isn’t the same opportunity to pass references, using declaratives or imports via the command line, as there is with csi.exe. Instead, the C# Interactive Window loads its default execution context from C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE\PrivateAssemblies\CSharpInteractive.rsp, which identifies the assemblies to reference by default:

```
# This file contains command-line options that the C# REPL
# will process as part of every compilation, unless
# "/noconfig" option is specified in the reset command.
/r:System
/r:System.Core
/r:Microsoft.CSharp
/r:System.Data
/r:System.Data.DataSetExtensions
/r:System.Xml
/r:System.Xml.Linq
SeedUsings.csx
```

Furthermore, the CSharpInteractive.rsp file references a default C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE\PrivateAssemblies\SeedUsings.csx file:

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

MARCH 7 – 11, 2016
BALLY'S HOTEL & CASINO, LAS VEGAS, NV

REAL TOOLS FOR TODAY'S IT CHALLENGES

Join us for TechMentor, March 7 - 11, 2016, as we make our return to fabulous Las Vegas, NV.

What sets TechMentor apart, and makes it a viable alternative to huge, first-party conferences, is the immediately usable IT education, providing the tools you need today while preparing you for tomorrow. Zero marketing-speak, a strong emphasis on doing more with the technology you already own, and solid coverage of what's just around the corner.

HOT TOPICS COVERED:

PowerShell
Security
DSC
Hyper-V
DevOps
Windows Server
Azure
And More!

TRACK TOPICS INCLUDE:



DataCenter



Client Devices



Dev Ops



IT Soft Skills



The Real Cloud

SAVE
\$400

REGISTER
BEFORE
JANUARY 13



USE PROMO CODE
TMJAN1

TECHMENTOREVENTS.COM/LASVEGAS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

The combination of these two files is why you can use `Console.WriteLine` and `Environment.CurrentDirectory` rather than the fully qualified `System.Console.WriteLine` and `System.Environment.CurrentDirectory`, respectively. In addition, referencing assemblies such as `Microsoft.CSharp` enables the use of language features like `dynamic` without anything further. (Modifying these files is how you'd change your "profile" or "preferences" so that the changes persist between sessions.)

More on the C# Script Syntax

One thing to keep in mind about the C# script syntax is that much of the ceremony important to a standard C# becomes appropriately optional in a C# script. For example, things like method bodies don't need to appear within a function and C# script functions can be declared outside of the confines of a class. You could, for example, define a NuGet Install function that appeared directly in the REPL window, as shown in **Figure 5**. Also, perhaps somewhat surprisingly, C# scripting doesn't support declaring namespaces. For example, you can't wrap the `Spell` class in a `Grammar` namespace: `namespace Grammar { class Spell {} }`.

It's important to note that you can declare the same construct (variable, class, function and so forth) over and over again. The last declaration shadows any earlier declaration.

Another important item to be conscious of is the behavior of the command-ending semicolon. Statements (variable assignments, for

The screenshot shows the Visual Studio C# Interactive window. The code in the window is:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

void Install(
    string packageId, string outputDirectory = null)
{
    if (outputDirectory == null)
    {
        outputDirectory = Environment.CurrentDirectory;
    }
    string arguments =
        $"install { packageId } "
        + $"-OutputDirectory \"{outputDirectory.Trim('')}\\""
        -NonInteractive";
}

// Assume nugget.exe is in the path.
ProcessStartInfo processInfo =
    new ProcessStartInfo("nugget.exe", arguments);

processInfo.UseShellExecute = false;
processInfo.Arguments = true;
```

A tooltip is displayed over the `processInfo.Arguments` line, showing the type as `string ProcessStartInfo.Arguments { get; set; }`. The cursor is positioned at the end of the line, just before the closing brace of the `Install` function.

Figure 5 Declaring a C# Script Function Outside of a Class Using the Visual Studio C# Interactive Window

example) require a semicolon. Without the semicolon the REPL window will continue to prompt (via a period) for more input until the semicolon is entered. Expressions, on the other hand, will execute without the semicolon. Hence, `System.Diagnostics.Process.Start("notepad")` will launch Notepad even without the ending semicolon. Furthermore, because the `Start` method call returns a process, string output of the expression will appear on the command line: `[System.Diagnostics.Process (Notepad)]`. Closing an expression with a semicolon, however, hides the output. Invoking `Start` with an ending semicolon, therefore, won't produce any output, even though Notepad will still launch. Of course, `Console.WriteLine("It would take a miracle.");` will still output the text, even with the semicolon, because the method itself is displaying the output (not the return from the method).

The distinction between expressions and statements can result in subtle differences at times.

The distinction between expressions and statements can result in subtle differences at times. For example, the statement `string text = "There's a shortage of perfect b....";` will result in no output, but `text="Stop that rhyming and I mean it"` will return the assigned string (because the assignment returns the value assigned and there's no semicolon to suppress the output).

The C# script directives for referencing additional assemblies (`#r`) and importing existing C# scripts (`#load`) are wonderful additions. (One could imagine complex solutions like `project.json` files to achieve the same thing that would not be as elegant.) Unfortunately, at the time of this writing, NuGet packages aren't supported. To reference a file from NuGet requires installing the package to a directory and then referencing the specific DLL via the `#r` directive. (Microsoft assures me this is coming.)

Note that at this time the directives refer to specific files. You can't, for example, specify a variable in the directive. While you'd expect this with a directive, it prevents the possibility of dynamically loading an assembly. For example, you could dynamically invoke "nuget.exe install" to extract an assembly (again see **Figure 5**). Doing so, however, doesn't allow your CSX file to dynamically bind to the extracted NuGet package because there's no way to dynamically pass the assembly path to the `#r` directive.

A C# CLI

I confess I have a love-hate relationship with Windows PowerShell. I love the convenience of having the Microsoft .NET Framework on the command line and

Figure 6 Keyboard Shortcuts for the C# Interactive Window

Enter	If the current submission appears to be complete, evaluate it. Otherwise, insert a new line.
Ctrl+Enter	Within the current submission, evaluate the current submission.
Shift+Enter	Insert a new line.
Escape	Clear the current submission.
Alt+UpArrow	Replace the current submission with a previous submission.
Alt+DownArrow	Replace the current submission with a subsequent submission (after having previously navigated backward).
Ctrl+Alt+UpArrow	Replace the current submission with a previous submission beginning with the same text.
Ctrl+Alt+DownArrow	Replace the current submission with a subsequent submission beginning with the same text (after having previously navigated backward).
UpArrow	At the end of the current submission, replace the current submission with a previous submission. Elsewhere, move the cursor up one line.
DownArrow	At the end of the current submission, replace the current submission with a subsequent submission (after having previously navigated backward). Elsewhere, move the cursor down one line.
Ctrl+K, Ctrl+Enter	Paste the selection at the end of interactive buffer, leave caret at the end of input.
Ctrl+E, Ctrl+Enter	Paste and execute the selection before any pending input in the interactive buffer.
Ctrl+A	First press, select the submission containing the cursor. Second press, select all text in the window.

the possibility of passing .NET objects across the pipe, rather than the traditional text of so many of the CLIs that came before. That said, when it comes to the C# language, I am partisan—I love its elegance and power. (To this day, I’m still impressed with the language extensions that made LINQ possible.) Therefore, the idea that I could have the breadth of Windows PowerShell .NET combined with the elegance of the C# language meant I approached the C# REPL as a replacement for Windows PowerShell. After launching csi.exe, I immediately tried commands like cd, dir, ls, pwd, cls, alias and the like. Suffice it to say, I was disappointed because none of them

The C# REPL provides a means of coding up short snippets or program units you can noodle on until they’re ready to be cut and pasted into larger programs.

worked. After pondering the experience and discussing it with the C# team, I realized that replacing Windows PowerShell was not what the team was focused on in version 1. Furthermore, it is the .NET Framework and, therefore, it supports extensibility both by adding your own functions for the preceding commands and even by updating the C# script implementation on Roslyn. I immediately set about defining functions for such commands. The start of such a library is available for download on GitHub at github.com/CSScriptEx.

For those of you looking for a more functional C# CLI that supports the previous command list out of the box now, consider

ScriptCS at scriptcs.net (also on GitHub at github.com/scriptcs). It, too, leverages Roslyn, and includes alias, cd, clear, cwd, exit, help, install, references, reset, scriptpacks, usings and vars. Note that, with ScriptCS, the command prefix today is a colon (as in :reset) rather than a number sign (as in #reset). As an added bonus, ScriptCS also adds support for CSX files, in the form of colorization and IntelliSense, to Visual Studio Code.

Wrapping Up

At least for now, the purpose of the C# REPL interface is not to replace Windows PowerShell or even cmd.exe. To approach it as such at the start will result in disappointment. Rather, I suggest you approach C# scripting and the REPL CLIs as lightweight replace-

ments for Visual Studio | New Project: UnitTestProject105 or the similarly purposed dotnetfiddle.net. These are C# and .NET targeted ways to increase your understanding of the language and .NET APIs. The C# REPL provides a means of coding up short snippets or program units you can noodle on until they’re ready to be cut and pasted into larger programs. It allows you to write more extensive scripts whose syntax is validated (even for little things like casing mismatches) as you write the code, rather than forcing you to execute the script only to discover you mistyped something. Once you understand its place, C# scripting and its interactive windows become a pleasure, the tool you’ve been looking for since version 1.0.

As interesting as C# REPL and C# scripting are on their own, consider that they also provide a stepping stone to being an extension framework for your own application—à la Visual Basic for Applications (VBA). With an interactive window and C# scripting support, you can imagine a world—not too far off—in which you can add .NET “macros” into your own applications again—without inventing a custom language, parser and editor. Now that would be a legacy COM feature worth bringing to the modern world none too soon. ■

MARK MICHAELIS is founder of IntelliTect, where he serves as its chief technical architect and trainer. For nearly two decades he has been a Microsoft MVP, and a Microsoft Regional Director since 2007. Michaelis serves on several Microsoft software design review teams, including C#, Microsoft Azure, SharePoint and Visual Studio ALM. He speaks at developer conferences and has written numerous books including his most recent, “Essential C# 6.0 (5th Edition)” (itl.tc/EssentialCSharp). Contact him on Facebook at facebook.com/Mark.Michaelis, on his blog at IntelliTect.com/Mark, on Twitter: @markmichaelis or via e-mail at mark@IntelliTect.com.

THANKS to the following Microsoft technical expert for reviewing this article:
Kevin Bost and Kasey Uhlenhuth

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

CODING Las Vegas

WE CAN BELIEVE IN



Visual Studio®
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE!



VISUAL STUDIO LIVE! is on a Campaign for Code in 2016, in support of developer education. First up on the campaign trail? Fabulous Las Vegas, where developers, software architects, engineers, designers and more will convene for five days of unbiased and cutting-edge education on the Microsoft Platform. Sharpen your skills in Visual Studio, ASP.NET, JavaScript, HTML5, Mobile, Database Analytics, and so much more.

EVENT PARTNERS



Magenic

SUPPORTED BY



msdn
magazine

Visual Studio
MAGAZINE

PRODUCED BY

105 MEDIA®
YOUR GROWTH. OUR BUSINESS.

LAS VEGAS • MARCH 7-11, 2016

BALLY'S HOTEL & CASINO



Turn the page
for more
Event details

CONNECT WITH VISUAL STUDIO LIVE!



[@VSLive](https://twitter.com/vslive)



[facebook.com – Search "VSLive"](https://facebook.com/Search%20%22VSLive%22)



[linkedin.com – Join the "Visual Studio Live" group!](https://linkedin.com/Join%20the%20%22Visual%20Studio%20Live%22%20group!)

DEVELOPMENT TOPICS INCLUDE:

- ALM/DevOps
- ASP.NET
- Cloud Computing
- Database & Analytics
- JavaScript / HTML5 Client
- Mobile Client
- Modern App Development
- UX/Design
- Visual Studio / .NET
- Windows Client

Be a responsible dev citizen. Register to code with us today!

**REGISTER BY
JANUARY 20
AND SAVE \$400!**



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

USE PROMO CODE **VSLJAN4**

VSLIVE.COM/LASVEGAS

Bally's Hotel & Casino
will play host to Visual Studio Live!, and is offering a special reduced room rate to conference attendees.



CONNECT WITH VISUAL STUDIO LIVE!

- [@VSLive](https://twitter.com/vslive)
- [Search "VSLive"](https://facebook.com)
- [Join the "Visual Studio Live" group!](https://linkedin.com)



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

ALM / DevOps		ASP.NET		Cloud Computing		Database and Analytics			
Pre-Conference Workshops: Monday, March 7, 2016 (Separate entry fee required)									
7:30 AM	9:00 AM								
		Pre-Conference Workshop Registration - Coffee and Morning Pastries							
9:00 AM	1:00 PM								
		M01 Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - <i>Miguel Castro</i>							
1:00 PM	2:00 PM								
		Lunch @ Le Village Buffet, Paris Las Vegas							
2:00 PM	6:00 PM								
		M01 Workshop Continues							
7:00 PM	9:00 PM								
		Dine-A-Round							
Day 1: Tuesday, March 8, 2016									
7:00 AM	8:00 AM								
		Registration - Coffee and Morning Pastries							
8:00 AM	9:00 AM								
		Keynote: To Be Announced							
9:15 AM	10:30 AM								
		T01 This session is sequestered, details will be released soon				T02 Angular 101 - <i>Deborah Kurata</i>			
10:45 AM	12:00 PM								
		T06 This session is sequestered, details will be released soon				T07 Angular 2.0: A Comparison - <i>Deborah Kurata</i>			
12:00 PM	1:00 PM					Lunch			
1:00 PM	1:30 PM					Dessert Break - Visit Exhibitors			
1:30 PM	2:45 PM					T11 Native Mobile App Development for iOS, Android and Windows Using C# - <i>Marcel de Vries</i>			
3:00 PM	4:15 PM					T12 ASP.NET 5 in All its Glory - <i>Adam Tuliper</i>			
3:00 PM	4:15 PM					T16 Getting Started with Hybrid Mobile Apps with Visual Studio Tools for Cordova - <i>Brian Noyes</i>			
4:15 PM	5:30 PM					T17 MVC 6 - The New Unified Web Programming Model - <i>Marcel de Vries</i>			
		Welcome Reception							
Day 2: Wednesday, March 9, 2016									
7:30 AM	8:00 AM								
		Registration - Coffee and Morning Pastries							
8:00 AM	9:15 AM								
		W01 Busy .NET Developer's Guide to Native iOS - <i>Ted Neward</i>				W02 Getting Started with Aurelia - <i>Brian Noyes</i>			
9:30 AM	10:45 AM					W06 Busy Developer's Guide to Mobile HTML/JS Apps - <i>Ted Neward</i>			
11:00 AM	12:00 PM					General Session: To Be Announced			
12:00 PM	1:00 PM					Birds-of-a-Feather Lunch			
1:00 PM	1:30 PM					Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)			
1:30 PM	2:45 PM					W11 Optimizing Applications for Performance Using the Xamarin Platform - <i>Kevin Ford</i>			
3:00 PM	4:15 PM					W16 Leverage Azure App Services Mobile Apps to Accelerate Your Mobile App Development - <i>Brian Noyes</i>			
4:30 PM	5:45 PM					W21 Building Cross-Platform Apps Using CSLA.NET - <i>Rockford Lhotka</i>			
7:00 PM	9:00 PM					Evening Out Event			
Day 3: Thursday, March 10, 2016									
7:30 AM	8:00 AM								
		Registration - Coffee and Morning Pastries							
8:00 AM	9:15 AM								
		TH01 Building for the Internet of Things: Hardware, Sensors & the Cloud - <i>Nick Landry</i>				TH02 Responsive Web Design with ASP.NET 5 - <i>Robert Boedigheimer</i>			
9:30 AM	10:45 AM					TH06 User Experience Case Studies - Good and Bad - <i>Billy Hollis</i>			
11:00 AM	12:15 PM					TH11 Pretty, Yet Powerful, How Data Visualization Transforms The Way We Comprehend Information - <i>Walt Ritscher</i>			
12:15 PM	1:45 PM					Lunch @ Le Village Buffet, Paris Las Vegas			
1:45 PM	3:00 PM					TH16 Exposing an Extensibility API for your Applications - <i>Miguel Castro</i>			
3:15 PM	4:30 PM					TH21 UWP Development for WPF and Silverlight Veterans - <i>Walt Ritscher</i>			
		TH22 Increase Website Performance and Search with Lucene.NET Indexing - <i>Ben Hoelting</i>							
Post-Conference Workshops: Friday, March 11, 2016 (Separate entry fee required)									
7:30 AM	8:00 AM								
		Post-Conference Workshop Registration - Coffee and Morning Pastries							
8:00 AM	12:00 PM								
		F01 Workshop: Upgrading Your Skills to ASP.NET 5 - <i>Mark Michaelis</i>				DETAILS COMING SOON! These sessions have been sequestered by our conference chairs. Be sure to check vslive.com/lasvegas for session updates!			
12:00 PM	1:00 PM					Lunch			
1:00 PM	5:00 PM					F01 Workshop Continues			





Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with

Magenic

BONUS CONTENT! Modern Apps Live! is now a part of Visual Studio Live! Las Vegas at no additional cost!

JavaScript /
HTML5 Client

Mobile
Client

UX/Design

Visual Studio /
.NET

Windows
Client

Modern Apps Live!

Pre-Conference Workshops: Monday, March 7, 2016 (Separate entry fee required)

Pre-Conference Workshop Registration - Coffee and Morning Pastries

M02 Workshop: DevOps in a Day - Brian Randell

M03 Workshop: SQL Server for Developers - Leonard Lobel

M04 Workshop: Modern App Technology Overview - Android, iOS, Cloud, and Mobile Web - Allen Conway, Brent Edwards, Kevin Ford & Nick Landry

Lunch @ Le Village Buffet, Paris Las Vegas

M02 Workshop Continues

M03 Workshop Continues

M04 Workshop Continues

Dine-A-Round

Day 1: Tuesday, March 8, 2016

Registration - Coffee and Morning Pastries

Keynote: To Be Announced

T03 Introduction to Next Generation of Azure Compute - Service Fabric and Containers - Vishwas Lele

T04 Developer Productivity in Visual Studio 2015 - Robert Green

T05 Defining Modern App Development - Rockford Lhotka

T08 Docker and Azure - Steve Lasker

T09 Building Windows 10 Line of Business Applications - Robert Green

T10 Modern App Architecture - Brent Edwards

Lunch

Dessert Break - Visit Exhibitors

T13 This session is sequestered, details will be released soon

T14 Exploring T-SQL Enhancements: Windowing and More - Leonard Lobel

T15 ALM with Visual Studio Online and Git - Brian Randell

T18 This session is sequestered, details will be released soon

T19 Geospatial Data Types in SQL Server - Leonard Lobel

T20 DevOps and Modern Applications - Dan Nordquist

Welcome Reception

Day 2: Wednesday, March 9, 2016

Registration - Coffee and Morning Pastries

W03 Exploring Microservices in a Microsoft Landscape - Marcel de Vries

W04 This session is sequestered, details will be released soon

W05 Reusing Logic Across Platforms - Kevin Ford

W08 Breaking Down Walls with Modern Identity - Eric D. Boyd

W09 JSON and SQL Server, Finally Together - Steve Hughes

W10 Coding for Quality and Maintainability - Jason Bock

General Session: To Be Announced

Birds-of-a-Feather Lunch

Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)

W13 Real-world Azure DevOps - Brian Randell

W14 Using Hive and Hive ODBC with HDInsight and Power BI - Steve Hughes

W15 Start Thinking Like a Designer - Anthony Handley

W18 This session is sequestered, details will be released soon

W19 Introduction to Spark for C# Developers - James McCaffrey

W20 Applied UX: iOS, Android, Windows - Anthony Handley

W23 Managing Windows Azure with PowerShell - Mark Michaelis

W24 Introduction to R for C# Programmers - James McCaffrey

W25 Leveraging Azure Services - Brent Edwards

Evening Out Event

Day 3: Thursday, March 10, 2016

Registration - Coffee and Morning Pastries

TH03 Unit Testing & Test-Driven Development (TDD) for Mere Mortals - Benjamin Day

TH04 Effective Agile Software Requirements - Richard Hundhausen

TH05 Building for the Modern Web with JavaScript Applications - Allen Conway

TH08 Unit Testing JavaScript - Ben Dewey

TH09 Lessons Learned: Being Agile in a Waterfall World - Philip Japikse

TH10 Building a Modern Android App with Xamarin - Kevin Ford

TH13 End-to-End Dependency Injection & Writing Testable Software - Miguel Castro

TH14 Real World Scrum with Team Foundation Server 2015 & Visual Studio Online - Benjamin Day

TH15 Building a Modern Windows 10 Universal App - Nick Landry

Lunch @ Le Village Buffet, Paris Las Vegas

TH18 Async Patterns for .NET Development - Ben Dewey

TH19 DevOps vs. ALM Different Measures of Success - Mike Douglas

TH20 Panel: Decoding Mobile Technologies - Rockford Lhotka

TH23 Improving Quality for Agile Projects Through Manual and Automated UI Testing - NO CODING REQUIRED! - Mike Douglas

TH24 This session is sequestered, details will be released soon

TH25 Analyzing Results with Power BI - Scott Diehl

Post-Conference Workshops: Friday, March 11, 2016 (Separate entry fee required)

Post-Conference Workshop Registration - Coffee and Morning Pastries

F02 Workshop: Building Business Apps on the Universal Windows Platform - Billy Hollis

F03 Workshop: Creating Awesome 2D & 3D Games and Experiences with Unity - Adam Tuliper

F04 Workshop: Modern Development Deep Dive - Jason Bock, Allen Conway, Brent Edwards & Kevin Ford

Lunch

F02 Workshop Continues

F03 Workshop Continues

F04 Workshop Continues

VSLIVE.COM/LASVEGAS



Moving Forward, Looking Back

We've just passed the winter solstice here in the northern hemisphere. The days are dark and cold, but they've started getting longer, albeit minutely. This is the month named for Janus, the Romans' two-faced god who looks both forward and back. This time of year and this god have lessons for us geeks.

In our industry, we're constantly looking forward. What's new today? What's in beta for release next quarter? What has Microsoft announced for next year? We rarely look back.

But what happens to programs written with last year's whiz-bang tools, or with the tools from the year before? These legacies often remain in service longer than we expect or want. As Y2K taught us, they continually need maintenance, bug fixes, adjustments to external changes. But as their technology ages, finding someone able and willing to work on them gets increasingly difficult.

I recently read that Larry Zottarelli, the last original programmer who worked on the Voyager 1 and 2 spacecraft, is retiring. NASA's Jet Propulsion Laboratory (JPL) will need a new programmer for some very old code.

I remember the Voyagers' launch in 1977, when the planets aligned for a grand tour. I remember marveling at some of their discoveries, such as volcanoes on Jupiter's moon Io, and the moon Prometheus that shepherds the F Ring of Saturn.

Voyager 1 has left the solar system and entered interstellar space. Voyager 2 will follow soon after this column runs. They're both still sending back data, though their radio waves now require 17 hours to reach us. Their plutonium thermoelectric generators should last another decade. What will they find? What will they tell us? No one knows, but I bet they'll need some changes to their software.

That means that JPL needs a geek to write them. I'm tempted to brush off my rusty FORTRAN and apply. Could I become a steely eyed missile man?

The JPL won't be getting a new, young guy. "[Voyager] was state of the art in 1975, but that's basically 40 years old," said Suzanne Dodd, JPL's manager of the Voyager program, in an October 2015 online *Popular Mechanics* article (bit.ly/10f9FuW). "Although some people can program in assembly language and understand the intricacy of the spacecraft, most younger people can't or really don't want to."

We see the same progression in the Windows world. So many different technologies have blazed like a comet into the sunshine of a Microsoft Professional Developers Conference. Their tails light

up the heavens for a few years, then fade as newcomers shoulder them aside. But their durable nuclei persist even as they recede into darkness. (For the ultimate example, see my columns on Visual Basic 6, at msdn.com/magazine/jj133828 and msdn.com/magazine/dn745870.)

Remember how the entire Microsoft world was neck-deep in COM for about 10 years? I wrote four books and many *MSDN Magazine* and *Microsoft Systems Journal* articles about it. COM is still used, or perhaps used again, for gluing together some of the internal components of Windows 10. But my .NET students have barely heard of it, let alone programmed it. When I assign them to access a COM server from their .NET program, they click the wizard buttons as the Microsoft documentation instructs them to, but without understanding what the system is doing. They're helpless when they hit my booby trap: a COM server written in VB6, but without the VB runtime DLL on which it depends. ("Error code 0x80004005 : Operation Failed.") It takes the starch out of them very quickly, tearing them down so that I can start building them back up in my own geeky image. (Student: "Platt, you're a sadistic bastard." Me: "Um, yeah, what's your point?")

I am currently consulting on a project requiring an archaeological dig through multiple generations of software: It's the .NET Framework on the surface, but has a lot of COM underneath, some implemented via the Microsoft Foundation Class Library and some via raw C++ (which isn't supposed to matter, but does). It has raw Win32 code in certain places. It uses .NET Remoting for communication. Few developers have the breadth of experience to tackle such a project. The pool shrinks every day as older ones retire or die, and the younger ones start with Microsoft Azure and never look back.

I'm starting a company to help clients with this type of project. I'm calling it Graybeard Software, at GrayBeardSoftware.com. (OldFartSoftware.com was taken.) Ping me if you need this kind of help, or if you can help me help others. If I'm not steering Voyager toward the Oort cloud, that is. ■

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.

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

FREE COMMUNITY LICENSE

A \$9,975 VALUE FOR FREE | SUPPORT AND UPDATES INCLUDED

CLAIM YOUR LICENSE

syncfusion.com/MSDNcommunity

Comprehensive offering includes more than 650 components across 13 platforms, an easy-to-use big data platform, and much more.

WEB	MOBILE	DESKTOP	FILE FORMATS	DATA SCIENCE	ENTERPRISE SOLUTIONS
					
ASP.NET	Android	Windows Forms	Excel	Big Data Platform	Dashboard Platform
ASP.NET MVC	HTML5/ JavaScript	WPF	Word	Predictive Analytics	Report Server
HTML5/ JavaScript	iOS	Universal Windows Platform	PDF		
Silverlight	Universal Windows Platform		PowerPoint		
LightSwitch	Windows Phone				
	WinRT				
	Xamarin				