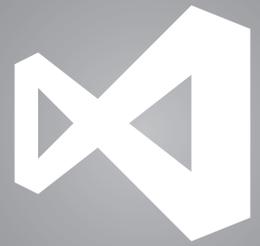


msdn magazine



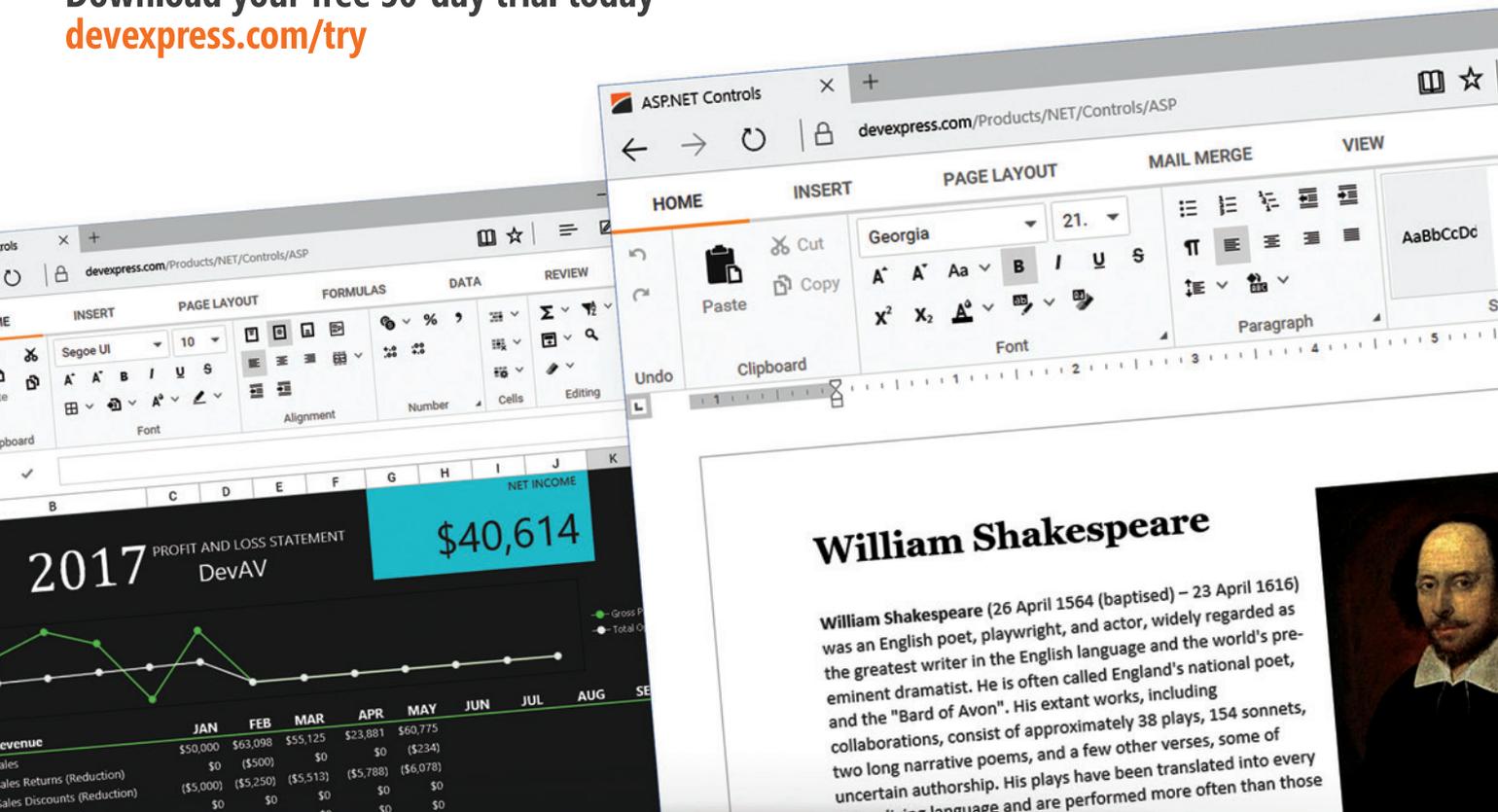
Visual Studio Extensions.....8

Office-Inspired ASP.NET & MVC Controls



Create high-impact line-of-business applications for the web with the DevExpress ASP.NET Subscription.

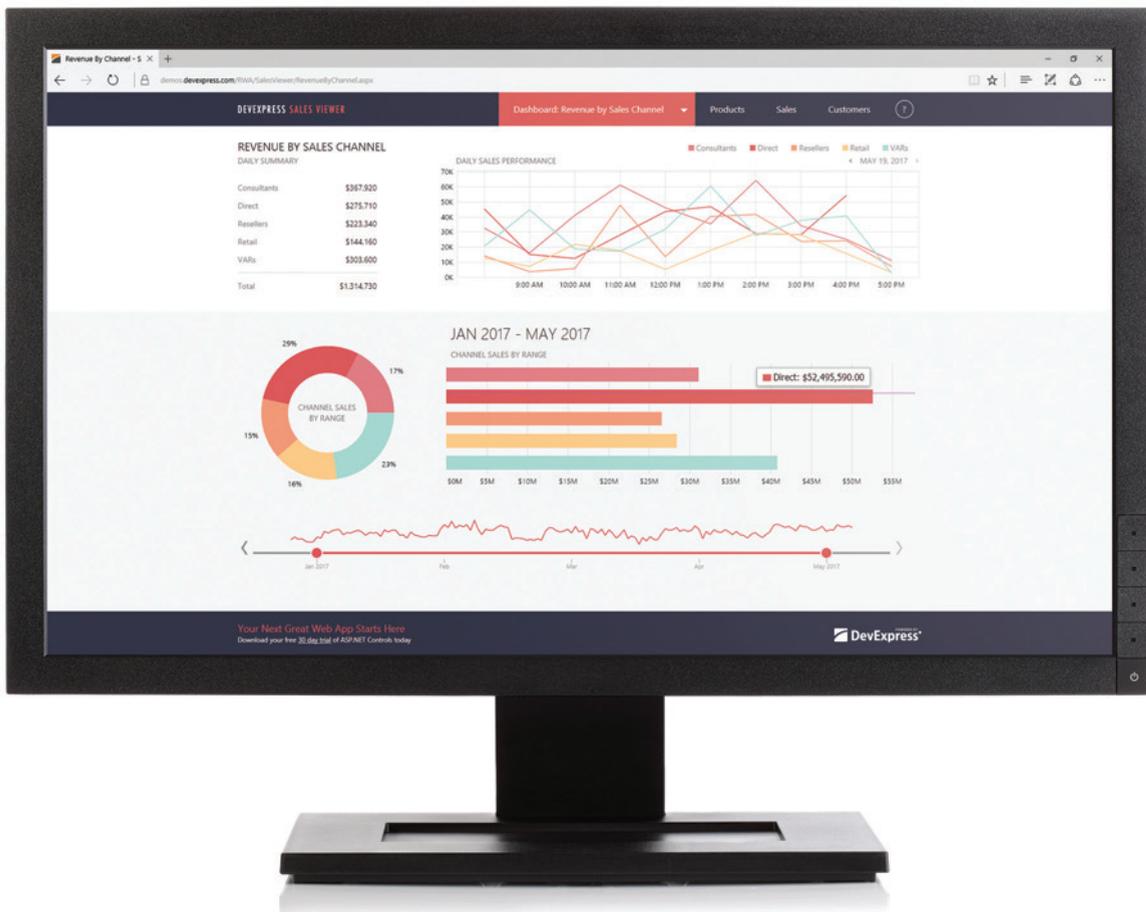
Download your free 30-day trial today
devexpress.com/try





Your Next Great Web App Starts Here

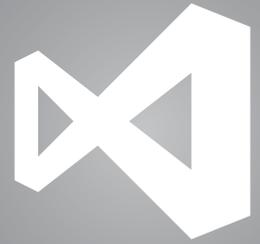
From apps that replicate the look and feel of Microsoft Office® 365, to high-impact decision support systems for your enterprise, DevExpress Web Controls for ASP.NET will help you build your best, without limits or compromise.



Download your free 30-day trial
an experience the DevExpress difference today.

devexpress.com/try

msdn magazine



Visual Studio Extensions.....8

Creating Extensions for Multiple Visual Studio Versions Carlos Quintero	8
How Xamarin.Forms Customization Took an FAA Drone App Higher Dan Hermes	18
Git Internals: Architecture and Index Files Jonathan Waldman	26
Actionable Messages for Outlook Woon Kiat Wong	34
Batch Processing Using a Serverless Architecture Joseph Fultz	44

COLUMNS

EDITOR'S NOTE

MEAN Machine
Michael Desmond, page 4

UPSTART

3 Demands: Mastering the Job Hunt
Krishnan Rangachari, page 6

TEST RUN

Deep Neural Network IO Using C#
James McCaffrey, page 58

THE WORKING PROGRAMMER

How To Be MEAN: Up-Angular-izing
Ted Neward, page 66

ESSENTIAL .NET

C# 7.0: Tuples Explained
Mark Michaelis, page 72

DON'T GET ME STARTED

Salt and Pepper
David Platt, page 80



Write Fast, Run Fast

with **Infragistics Ultimate** Developer Toolkit

Includes 100+ beautiful, fast grids, charts, and other UI controls, plus productivity tools for quickly building high-performing web, mobile, and desktop apps

Featuring

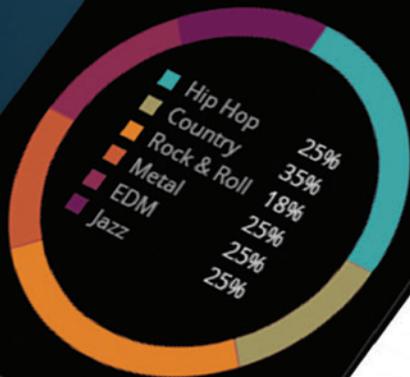
- Xamarin UI controls with innovative, code-generating productivity tools
- JavaScript/HTML5 and ASP.NET MVC components, with support for:



Also includes controls for WPF, Windows Forms, and ASP.NET, plus prototyping, remote usability testing, and more.

Get started today with a free trial,
reference apps, tutorials, and eBooks at
[Infragistics.com/Ultimate](https://www.infragistics.com/Ultimate)

MUSIC BY GENRE



FINANCE

Example Corporation (EXMPL)

23.18 ↓ -0.34 (-0.72%)
2/15/2017, 9:55 AM

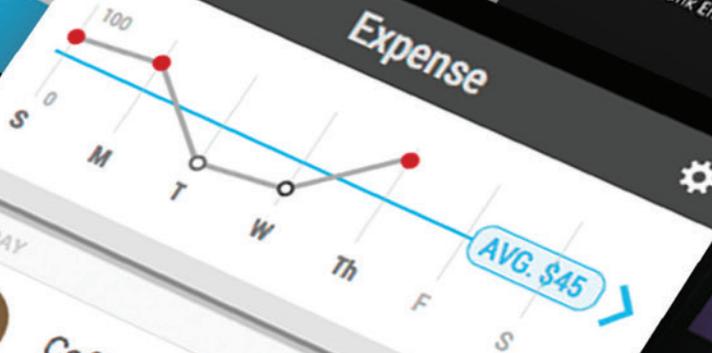


Close	Date	High
22.34	Thu Jan 19 2017	22.6294
22.288	Wed	
21.9729	Tue Jan 17 2017	
22.12	Mon	
21.7834	Sun Jan 15 2017	

1 - 5 of 10 records

JULY 29
WEEK 31

Expense



- TODAY**
 - Coffee @ STARBUCKS \$19.25
 - Movie RISE OF GORT 4.75
- YESTERDAY**
 - Travel GORT-A 4.75

Mail App interface showing a list of email items and a rich text editor. The list includes items like 'File', 'Active', 'Tab Item', and 'Item Hover'. The editor shows text like 'Name LastName' and 'This is the subject of the email'.

General Manager Jeff Sandquist
Director Dan Fernandez
Editorial Director Mohammad Al-Sabt mmeditor@microsoft.com
Site Manager Kent Sharkey
Editorial Director, Enterprise Computing Group Scott Bekker
Editor in Chief Michael Desmond
Features Editor Sharon Terdeman
Features Editor Ed Zintel
Group Managing Editor Wendy Hernandez
Senior Contributing Editor Dr. James McCaffrey
Contributing Editors Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt
Vice President, Art and Brand Design Scott Shultz
Art Director Joshua Gould



President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

ART STAFF

Creative Director Jeffrey Langkau
Associate Creative Director Scott Rovin
Senior Art Director Deirdre Hoffman
Art Director Michele Singh
Art Director Chris Main
Senior Graphic Designer Alan Tao
Senior Web Designer Martin Peace

PRODUCTION STAFF

Print Production Coordinator Lee Alexander

ADVERTISING AND SALES

Chief Revenue Officer Dan LaBianca
Regional Sales Manager Christopher Kourtoglou
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
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 Bastionell
Senior Program Manager, Client Services & Webinar Production Chris Flack
Project Manager, Lead Generation Marketing Mahal Ramos

MARKETING

Chief Marketing Officer Carmel McDonagh
Vice President, Marketing Emily Jacobs
Marketing & Editorial Assistant Megan Burpo

ENTERPRISE COMPUTING GROUP EVENTS

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



Chief Executive Officer
Rajeev Kapur
Chief Operating Officer
Henry Allain
Chief Financial Officer
Craig Rucker
Chief Technology Officer
Erik A. Lindgren
Executive Vice President
Michael J. Valenti
Chairman of the Board
Jeffrey S. Klein

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

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

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

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

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

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

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

Reaching the Staff

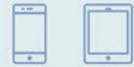
Staff may be reached via e-mail, telephone, fax, or mail. E-mail: To e-mail any member of the staff, please use the following form: FirstInitialLastname@1105media.com
Irvine Office (weekdays, 9:00 a.m. - 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618
Corporate 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.





LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs



OCR FOR ANY CLIENT, DEVICE OR SERVER. DONE.



Fast, accurate and reliable optical character recognition for use in any application or environment

Utilize multiple cores for unparalleled performance

Supports multiple text recognition engines including OCR, ICR, MICR, MRZ and MRP

Automatically detect, segment and recognize multiple languages on the same document

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

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM



SALES@LEADTOOLS.COM
 800.637.1840





MEAN Machine

It was August 2015 when Ted Neward officially settled down. Neward for years has used his column, *The Working Programmer*, as a platform to cover everything and anything—from his 10-part opus on Multiparadigmatic .NET development, to his experiment with an ELIZA-like intelligent conversation bot (this back in 2012, mind you), to his work with alternative frameworks and databases like Oak, Cassandra and MongoDB. And don't even get me started on his brilliant LOLCODE column (msdn.com/magazine/dn166934)—I'm still chuckling about that.

But it was two years ago this month that Neward—accidentally, it turns out—settled on a single topic. Since his August 2015 column, “How To Be MEAN: Getting Started” (msdn.com/magazine/mt185576), Neward has been exploring the popular MEAN stack, consisting of MongoDB, Express, Angular and Node.js. That article was intended to be the first of perhaps half a dozen columns exploring MEAN. Now, 24 months later, Neward is still at it. And judging by the online traffic his columns are generating, he could go another 24 months.

“I originally thought this would maybe be a six- to nine-piece series, and then we'd move on to some other things,” Neward says. “In fact, I have a number of .NET-centric ideas waiting in the wings, including a piece or two on static code analyzers, which is about as far from the world of dynamic, typeless, JavaScript-y programming as you can get.”

Neward describes writing about the MEAN stack as “both tricky and rewarding,” with frequent, major updates unveiling green fields to explore. This was particularly true of the Angular 2 release in September 2016, which Neward describes as a “complete transformation,” but he also singles out changes to both TypeScript and the ECMAScript language. And while Neward says MEAN can

solve thorny problems that seem to bedevil other platforms, the bill eventually comes due.

“When you're deeper in, you discover that what was easy on your old platform—like .NET or JVM or whatever—is not so easy here, and you're forced to sit back in your chair and go, ‘Huh.’”

“I originally thought this would maybe be a six- to nine-piece series, and then we'd move on to some other things.”

Ted Neward, *MSDN Magazine* Columnist

At the end of the day, MEAN is just an architectural stack, much like LAMP (Linux, Apache, MySQL, PHP) before it, and Neward cautions against ascribing too much to it.

“We talked about this a year or so ago: You can build an ASP.NET WebAPI + CouchDB + ReactJS stack, and call it ‘ARC,’ if you like, and have just as much success with it as you can with MEAN.”

Neward should know. He's been having success with MEAN for two years now, and he's not done yet. In upcoming issues, he says he plans to dive into Angular Routing, and from there into testing and forms.

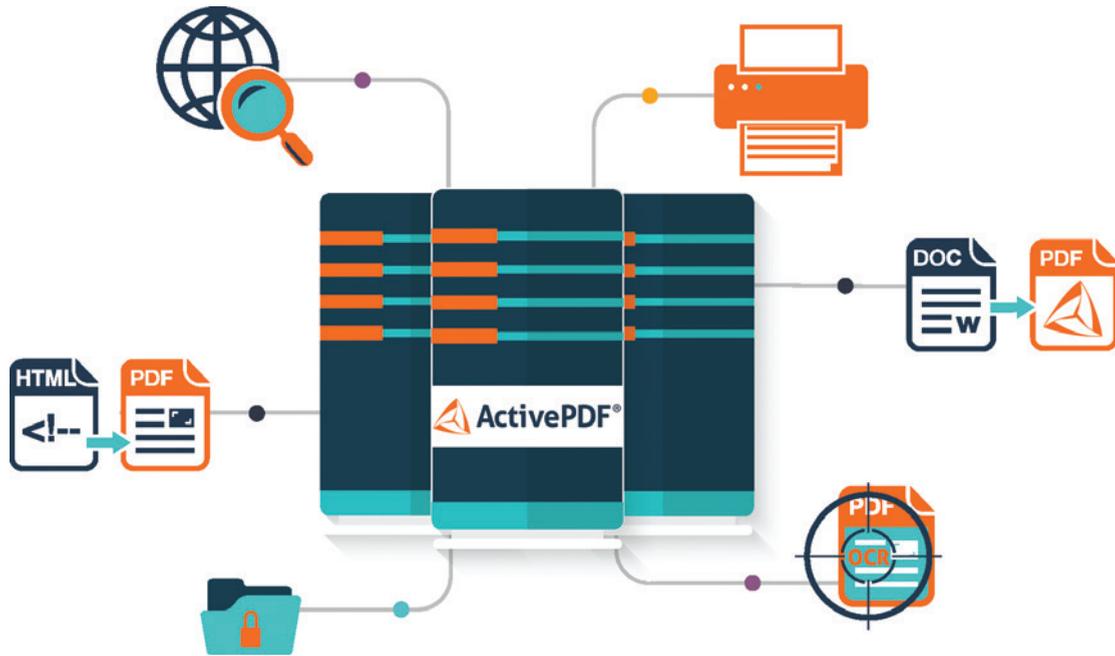
Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

© 2017 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.



ActivePDF provides developers and IT professionals with the capability to move closer to a fully functional digital environment. Digitalize your organization by integrating PDF manipulation and automation into your business process workflow. Join the PDF Revolution!

ActivePDF Family of Products

DocGenius™ Developer tools for creating software applications that have embedded PDF functionality.

Toolkit

Scalable & flexible PDF processing library

Xtractor

Search PDF to retrieve or extract text & images

WebGrabber

Server-Based HTML-to-PDF conversion

Server

PDF generation for legacy software

DocSight™ Unattended server applications that enable creation, conversion and other PDF functionality.

OCR

Image conversion of PDF files to searchable text

DocConverter

High-volume document conversion

Meridian

Network PDF printer for unlimited users

DocSpace™ Server applications with user interfaces for viewing, creating and interacting with PDF.

ReaderPlus

Browser-based PDF viewer and editor

Download your FREE 30-day trial →

www.ActivePDF.com

Toll Free US: 866 468 6733 | Outside US: +1 949 582 9002

3 Demands: Mastering the Job Hunt

When you're job hunting, there are three demands that a potential employer might make: skills, experience and achievements. The more you can exceed expectations on these demands, the more likely you'll get the job.

Skills

Maybe all your experience is in the Microsoft .NET Framework and Azure, but a job you're interested in is recruiting developers for Java and a competing cloud platform. What do you do?

First, about 30 percent to 40 percent of technology companies (including the likes of Facebook and Google) are technology-agnostic in their hiring. Even if you have zero experience in their primary stacks, they'll hire you if you're a good engineer; they'll trust you to learn the tools and languages quickly.

Second, another 20 percent to 30 percent of companies will give "credit" for similar-enough technologies. For example, some shops on a non-Azure cloud will "honor" your Azure experience, and some Java shops will look favorably on your .NET experience.

At the remaining 30 percent to 50 percent of companies—the ones that are looking for specific stack experience—developers make the mistake of trying to sell themselves as "quick learners." The problem is *everybody* claims to be a quick learner!

Instead, the trick is to gain and demonstrate *some* experience in the stack in which you have zero experience. Your day job may be in .NET, but you can take on skunkworks or tool projects at work using other technologies. Typically, you have more technical wiggle room with non-core projects. If you do this over a few months, you can shift your resume from being 100 percent C# projects to, say, 80 percent C# and 20 percent Python.

Then, in your interviews, if you get grilled on your stack experience, you can say, "My background is in C#. Over the last few months, I've been using Python more and more in my projects, and I like it a lot. One of the things that's attracting me to this role is the opportunity to use Python even more."

This comes off as not only sincere, but its vulnerable nature protects you. You aren't claiming to be a Python god or goddess, but you're also not bemoaning your Python newbie status. Plus, you've weaved a story of how your past experience leads into this future opportunity.

Experience

Sometimes, the whole point of a job switch is that you want a better position than the one you're in right now. In effect, you want to "up-level." But how do you upgrade from a manager position to a director one? How do you get hired as a manager if you've never managed people?

Upgrading from a team lead to a senior manager or director is relatively easy, especially if you're at a big company. You can demand a loftier title when you switch to a smaller company or a startup. The smaller company appreciates your big-company experience, maturity, expertise, and skillset, and you appreciate the upgraded title, the greater scope of your responsibilities, and the opportunity for more direct impact.

Now, if you have zero management experience and your goal is to become a manager at a new job, it's more challenging. That's a little bit like a 12-year-old you've never met asking to babysit your 1-year-old. The 12-year-old isn't a known commodity, and neither are non-managers who want to get hired as managers.

In such situations, it helps for you to become an "acting manager" at your current role. This means you can manage interns, mentor junior- and mid-level engineers, and act as a project manager on some projects, as technical lead on others, and as design architect on still others. This way, you don't have to ask the companies you interview with to trust your "potential." Instead, you can showcase your status as a manager in your responsibilities (if not in title) and share five to 10 experiences to prove it.

Achievements

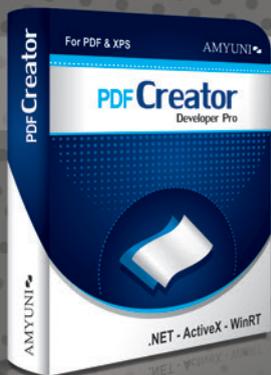
Sometimes, it may feel like the world is being flooded with new software engineers. How do you stand out in this virtually indistinguishable ocean?

While the number of software engineers keeps increasing, I observe—in my interactions with hundreds of software engineers each year—that there's still an extreme shortage of self-aware software engineers with strong communication skills and good technical chops.

So, the more you invest in yourself as an engineer, the more you stand out and live up to your own potential. This includes practicing mock technical interviews on sites like pramp.com, devouring technical interview books and courses, investing in your own coaching and personal development, and polishing your resume and story-telling skills. It also includes mastering how to position yourself in every stage of the job hunt.

Ultimately, you aren't competing with an army of mediocre software engineers; in fact, the more mediocre engineers there are, the easier it is to stand out! You're competing with yourself to set extraordinarily high standards of performance, and pursue them with determination, dedication and enthusiasm. ■

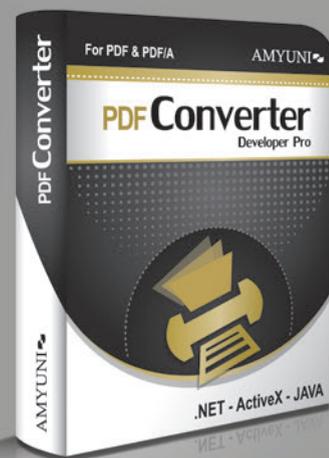
KRISHNAN RANGACHARI helps brilliant developers have amazing careers. Visit RadicalShifts.com for his free courses.



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

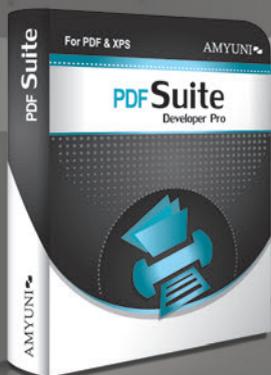


- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .NET and ActiveX/COM
- New Universal Apps DLLs enable publishing C#, C++, CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module



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

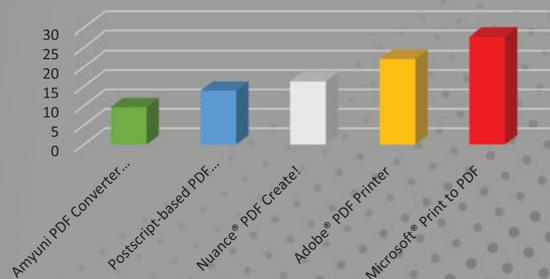


Complete Suite of Accurate PDF Components

- 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

Benchmark Testing - Amyuni vs Others

Seconds required to convert a document to PDF



Other Developer Components from Amyuni®

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



AMYUNI
Technologies

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

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

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

All development tools available at
www.amyuni.com

Creating Extensions for Multiple Visual Studio Versions

Carlos Quintero

The release of a new version of Visual Studio is always a challenge for developers of extensions (packages, add-ins, templates and so forth). For example, Visual Studio 2010 introduced the new Visual Studio Installer for eXtensions (VSIX files); Visual Studio 2012 introduced the light/dark themes; and Visual Studio 2015 removed add-ins (with the Add-In Manager); not to mention that each Visual Studio version provides a new SDK, new extensibility assemblies and new APIs. With Visual Studio 2017, this challenge is even bigger, due to its new modular setup based on workloads and individual components, and to a new version of the manifest for the VSIX deployment mechanism. While some developers (most notably from Microsoft) release a different new extension for each Visual Studio version, most would prefer to release a single updated extension that can target the widest range of Visual Studio versions.

This article discusses:

- How Visual Studio extensions are created, deployed and installed
- How to create a single package for multiple Visual Studio versions
- How to deploy a package with a single VSIX file

Technologies discussed:

Visual Studio 2012, 2013, 2015 and 2017,
Microsoft .NET Framework, VSIX files

Code download available at:

msdn.com/magazine/0817magcode

In this article, I'll show you how to accomplish this. For this purpose, I'll focus on the most common scenario: a package with a command, created in a managed language (C#, in this case) and deployed as a VSIX file.

The goals to be accomplished are the following:

- To use a single Visual Studio project to create the package.
- To use Visual Studio 2017 for development and debugging.
- To generate a single package DLL as the result of the build.
- To put that single DLL inside a single VSIX file.
- To be able to install that VSIX file on Visual Studio 2017 and on many past versions (2015, 2013 and so on).

Because two artifacts are needed—a DLL file (which is the package) and a VSIX file (which is the deployment vehicle for the package)—I'll explain each of these separately: First, how they work at installation or run time; second, how to develop them.

The VSIX File

As mentioned earlier, Visual Studio 2010 introduced the VSIX deployment mechanism to install Visual Studio extensions, and it's been the preferred way ever since. A VSIX file has the extension .vsix and can be installed in different ways. If the VSIX file is published on the Visual Studio Marketplace (formerly Visual Studio Gallery) and it's compatible with the Visual Studio version and edition you're using, you can install it using the Extensions and Updates dialog. Under the Tools menu, click on Extensions and Updates and then go to Online | Visual Studio Marketplace (see **Figure 1**).

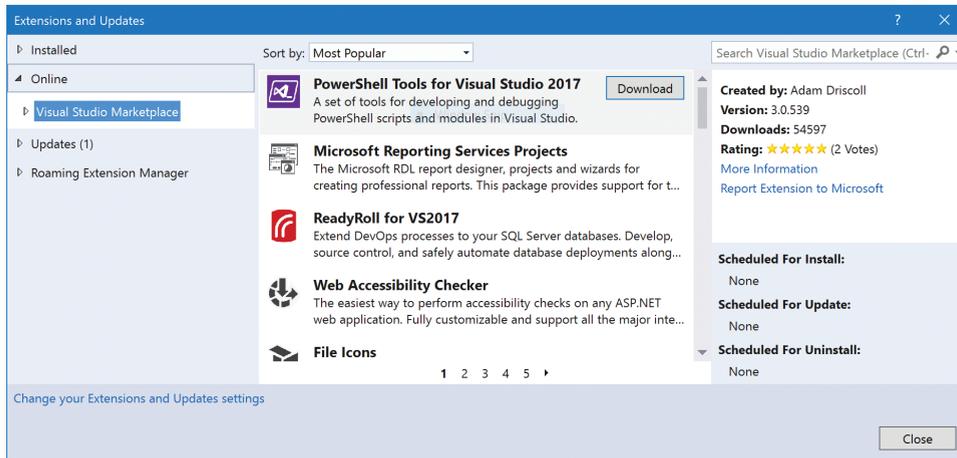


Figure 1 The Extensions and Updates Dialog Window

You can also double-click a VSIX file. When this happens, a Visual Studio Launcher (`C:\Program Files (x86)\Common Files\Microsoft Shared\MSEnv\VSLauncher.exe`) associated to the .vsix file extension is executed; this locates the VSIXInstaller.exe utility of the highest installed Visual Studio version (the highest version is required to be able to install to all lower versions). Then, the VSIX Installer shows the dialog in **Figure 2** so you can select the compatible Visual Studio versions and editions in which to install the extension.

VSIX files can be installed programmatically, too, using the VSIXInstaller.exe utility with its command-line options, such as the target Visual Studio version (2017, 2015 and so on) and edition (Community, Professional and the like). You can find that utility in the `Common7\IDE` subfolder of your Visual Studio installation.

In any case, either Visual Studio or the VSIXInstaller.exe utility needs to know which Visual Studio versions and editions the VSIX file supports. That information can be discovered via a manifest file inside the file. The VSIX file is actually a .zip file, so you can rename its .vsix file extension to .zip and then open it to examine its contents (see **Figure 3**).

As you can see, there are several files inside: The .dll file is the package DLL. The .pkgdef file is used at installation time to add some keys to the Windows Registry that allows Visual Studio to recognize the DLL as a package. The [Content_Types].xml file describes the content type for each file extension (.dll, .json and so forth). The catalog.json and manifest.json files are required by Visual Studio 2017. And the extension.vsixmanifest file describes the name of the extension, version, and more, and which Visual Studio versions and editions it supports.

You can unzip the extension.vsixmanifest file and open it with a text editor to examine its contents, which will look similar to what's shown in **Figure 4**.

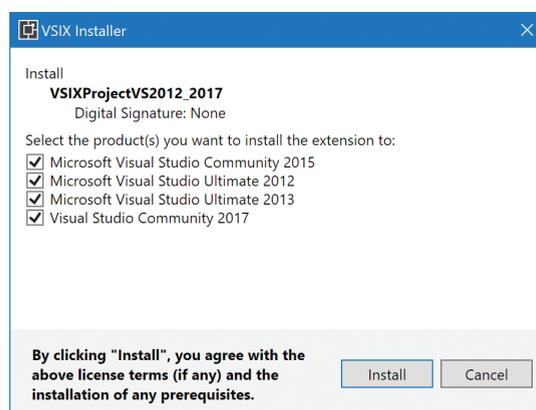


Figure 2 The VSIX Installer

As you can see, the manifest states in the InstallationTarget XML element the supported Visual Studio editions. Here, the Microsoft.VisualStudio.Pro value targets the Professional edition *and higher*, such as the Premium, Ultimate, Enterprise and any other such editions. Note that it also targets the Community edition, which is basically a Professional edition with some licensing restrictions and without some features. It also states the range of supported Visual Studio versions: 10.0 (2010), 11.0 (2012), 12.0 (2013), 14.0 (2015), 15.0 (2017).

When the VSIX file of a per-user extension is installed (either by Visual Studio or by the VSIX Installer), the files inside are unzipped and copied to a random folder at this location: `C:\Users\<user>\AppData\Local\Microsoft\VisualStudio\<version number>\Extensions\<random folder>`. The <version number> can have an “Exp” suffix appended for the “Experimental Instance” (explained later), and for Visual Studio 2017 it will also include the “instance id” of the installed Visual Studio. This instance id is randomly generated at Visual Studio install time; it was added to support side-by-side installations of different editions of the same version (2017) of Visual Studio, something that wasn't possible before. For machine-wide extensions, the subfolder `Common7\IDE\Extensions` is used. Notice that in any case each Visual Studio version uses its own folder for its extensions.

While it would be nice if all Visual Studio versions supported the same manifest format, unfortunately that's not the case. Visual Studio 2010 introduced VSIX and the first version of the manifest. Visual Studio 2012 introduced version 2, which is completely different and incompatible with version 1. However, Visual Studio 2012, 2013 and 2015—all of which support version 2—can still accept a version 1 manifest, so you can build a VSIX file with a version 1 and target from Visual Studio 2010 to Visual Studio 2015. But Visual Studio 2017 supports neither version 1 nor version 2. Instead, it requires a third version of the manifest. Fortunately, version 3 keeps using the value “2.0.0.0” in the

Version attribute of the PackageManifest XML element and it adds only an XML element named <Prerequisites> (and the two new files, catalog.json and manifest.json, into the VSIX file). So, it's completely backward-compatible with the second version, supported by Visual Studio 2012, 2013 and 2015 (but not by Visual Studio 2010, which only supports version 1). This means that you can't target Visual Studio 2010-2017 with a single VSIX file. From this point, I'll give up on Visual Studio 2010 and will continue with a VSIX file that supports Visual Studio 2012, 2013, 2015 and 2017.

Name	Type	Compressed size
[Content_Types].xml	XML File	1 KB
catalog.json	JSON File	1 KB
extension.vsixmanifest	Extension Manifest	1 KB
manifest.json	JSON File	1 KB
VSIXProjectVS2012_2017.dll	Application extension	428 KB
VSIXProjectVS2012_2017.pdb	Program Debug Database	16 KB
VSIXProjectVS2012_2017.pkgdef	Package Definition Registration File	1 KB

Figure 3 Contents of a VSIX File

The Package DLL

A managed Visual Studio package is a DLL that contains a class that inherits from `Microsoft.VisualStudio.Shell.Package`. It's decorated with certain attributes that help at build time to generate a `.pkgdef` file (which, as mentioned earlier, you can find inside the VSIX file and in the installation folder of the extension). The `.pkgdef` file is used at startup (older versions of Visual Studio) or at installation time (version 15.3 of Visual Studio 2017) to register the DLL as a package for Visual Studio. Once it's registered, Visual Studio will try to load the package at some point, either on startup or when one of its commands is executed if the package uses delay loading (which is the best practice). During the attempt to load the managed DLL and initialize the package, three things happen: the DLL will be loaded by the Common Language Runtime (CLR) of a Microsoft .NET Framework version; it will use some DLLs provided by a .NET Framework; and it will use some DLLs provided by Visual Studio. I will examine each of these in turn.

A .NET Framework is the sum of two things: The CLR + libraries (both base class and additional libraries). The CLR is the runtime (the JIT compiler, garbage collector and so forth) and it loads managed DLLs. In the distant past, each .NET Framework version 1.0, 1.1 and 2.0 (used by Visual Studio.NET 2002, Visual Studio.NET 2003 and Visual Studio 2005) provided its own CLR version (1.0, 1.1 and 2.0). However, the .NET Frameworks 3.0 and 3.5, used by Visual Studio 2008, continued to use the exact same CLR 2.0 of .NET Framework 2.0, instead of introducing a new one. Visual Studio 2010 introduced .NET Framework 4 and CLR 4.0, but since then all new .NET Frameworks 4.x have used CLR 4.0 (although swapping it "in-place" with a backward-compatible version rather than reusing the exact CLR 4.0 of .NET Framework 4). Since Visual Studio 2012 and higher

all use CLR 4.0, the CLR version is not a problem when the DLL of an extension targets Visual Studio 2012, 2013, 2015 and 2017.

Libraries constitute the second part of a .NET Framework; these are DLLs referenced by a Visual Studio project and used at run time. To develop a single extension that targets multiple versions of Visual Studio, you must use the highest .NET Framework installed by default by the lowest Visual Studio version that you want to target. This means that if you want to target Visual Studio 2012 and higher, you need to use .NET Framework 4.5. You can't use, say, .NET Framework 4.5.1 introduced by Visual Studio 2013, because any DLL introduced in that version would not be present on a computer with only Visual Studio 2012 installed. And unless you really need that DLL, you won't want to force such users to install .NET Framework 4.5.1 to use your extension (it could hurt sales or downloads and support).

The release of a new version of Visual Studio is always a challenge for developers of extensions.

The extension also needs DLLs that are provided by Visual Studio (typically named `Microsoft.VisualStudio.*`). At run time, Visual Studio finds its DLLs at some well-known locations, such as the folder `Common7\IDE` with its subfolders `Common7\IDE\PublicAssemblies` and `Common7\IDE\PrivateAssemblies`, and from the Global Assembly Cache (GAC). The GAC for .NET Framework 4.x is located at `C:\Windows\Microsoft.NET\assembly` (there's another GAC at `C:\Windows\assembly`, but that one is for older .NET Frameworks). Visual Studio 2017 uses a more isolated installation that avoids the GAC, relying instead on the folders described previously.

There are a couple of key principles to follow when developing and generating a VSIX file: You must use the versions provided by the lowest Visual Studio version your extension targets. That means that if you want to target Visual Studio 2012 and higher, you must use only assemblies and extensibility APIs provided by that ver-

```

<?xml version="1.0" encoding="utf-8" ?>
<PackageManifest Version="2.0.0" xmlns="http://schemas.microsoft.com/developer/vsx-schema/2011" xmlns:d="http://schemas.microsoft.com/
  <Metadata>
    <Identity Id="25a3e631-bb07-4945-a11b-cb9fae973726" Version="1.0" Language="en-US" Publisher="Carlos Quintero" />
    <DisplayName>VSIXProjectVS2012_2017</DisplayName>
    <Description xml:space="preserve">VSIX Project for VS 2012-2017</Description>
  </Metadata>
  <Installation>
    <InstallationTarget Id="Microsoft.VisualStudio.Pro" Version="[11.0,15.0]" />
  </Installation>
  <Dependencies>
    <Dependency Id="Microsoft.Framework.NDP" DisplayName="Microsoft .NET Framework" d:Source="Manual" Version="[4.5,)" />
  </Dependencies>
  <Prerequisites>
    <Prerequisite Id="Microsoft.VisualStudio.Component.CoreEditor" Version="[15.0,16.0]" DisplayName="Visual Studio core editor" />
  </Prerequisites>
  <Assets>
    <Asset Type="Microsoft.VisualStudio.VsPackage" d:Source="Project" d:ProjectName="%CurrentProject%" Path="|%CurrentProject%;Pkgde
  </Assets>
</PackageManifest>

```

Figure 4 The Contents of a Manifest File

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, AS4, EDI/X12, OFTP ...
- **Credit Card Processing**
Authorize.Net, ACH, 3-D Secure ...
- **Shipping & Tracking**
FedEx, UPS, USPS ...
- **Accounting & Banking**
QuickBooks, OFX, SAP ...
- **Internet Business**
Amazon, PayPal, Google ...
- **Internet Protocols**
FTP, SMTP, IMAP, POP, WebDav ...
- **Secure Connectivity**
SSH, SFTP, SSL, Certificates ...
- **Encryption & Certificates**
X.509, OpenPGP, SHA, S/MIME ...
- **Network Management**
SNMP, MIB, LDAP, Monitoring ...
- **Compression**
Zip, Gzip, Jar, AES, 7Zip ...



Our **Red Carpet Subscription** includes all product lines + updates for one year.

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. For more than 20 years, 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

sion (or lower). If your extension uses a DLL introduced by Visual Studio 2013 or higher, the extension won't work on a machine with only Visual Studio 2012. The second principle is that the extension never must deploy Visual Studio DLLs, neither to the locations I mentioned (folders of Visual Studio or GAC), nor to the installation folder of the extension. These DLLs are provided by the target Visual Studio, which means that the VSIX file shouldn't include them.

Many Visual Studio DLLs have a version number (8.0 ... 15.0) in the name, such as `Microsoft.VisualStudio.Shell.11.0.dll` or `Microsoft.VisualStudio.Shell.Immutable.10.0.dll`. These help to identify the Visual Studio version that introduced them, but don't get fooled: it's a name, not a version. For example, there are four versions (11.0.0.0, 12.0.0.0, 14.0.0.0 and 15.0.0.0) of `Microsoft.VisualStudio.Shell.11.0.dll`, each one provided, respectively, by a Visual Studio version (2012, 2013, 2015 and 2017). The first three 11.0.0.0 to 14.0.0.0 are installed by the respective Visual Studio version in the GAC and the fourth version, 15.0.0.0, used by Visual Studio 2017, is installed in the `Common\IDE\PrivateAssemblies` folder.

Because an extension that targets Visual Studio 2012 and higher must use Visual Studio assemblies with version 11.0.0.0 (the first principle mentioned earlier), this means that the reference `Microsoft.VisualStudio.Shell.11.0.dll` must be version 11.0.0.0. But because that version isn't installed by Visual Studio 2013 and higher (they start at version 12.0.0.0), and the extension shouldn't deploy Visual Studio DLLs (the second principle), wouldn't the extension fail when trying to use that Visual Studio DLL? The answer is no, and it's thanks to an assembly-binding redirection mechanism provided by the .NET Framework, which allows you to specify rules like "when something requests this version of an assembly, use this newer version of it." Of course, the new version must be fully backward-compatible with the old version. There are several ways to redirect assemblies from one version to another. One way is this: An executable (.exe file extension) can provide an accompanying configuration file (.exe.config file extension) that specifies the redirections. So, if you go to the `Common7\IDE` folder of your Visual Studio installation, you'll find the `devenv.exe` executable of Visual Studio, and a `devenv.exe.config` file. If you open the .config file with a text editor, you'll see that it contains lots of assembly redirections:

```
<dependentAssembly>
  <assemblyIdentity
    name="Microsoft.VisualStudio.Shell.11.0"
    publicKeyToken="b03f5f7f11d50a3a"
    culture="neutral"/>
  <bindingRedirect
    oldVersion="2.0.0.0-14.0.0.0"
    newVersion="15.0.0.0"/>
</dependentAssembly>
```

So, Visual Studio 2017 (15.0) has an assembly version redirection for `Microsoft.VisualStudio.Shell.11.0` that states that whenever something requests old versions 2.0.0.0 to 14.0.0.0, use the new version 15.0.0.0 instead. That's how Visual Studio 2013 or later can use an extension referencing `Microsoft.VisualStudio.Shell.11.0` version 11.0.0.0, even if they don't provide that exact version.

Developing the Extension

Now that you know how things work at run time, you can develop the package. To recap, you'll create a VSIX project using Visual

Studio 2017 with a manifest that targets Visual Studio versions from 12.0 to 15.0; it will contain a package and a command; and it will use only references with version 11.0.0.0 (or lower) installed by Visual Studio 2012.

You might wonder at this moment which Visual Studio versions should be installed on your development machine. The best practice is to have two development machines as follows: On the first, if you have enough space on your disk, install all the Visual Studio versions—2012, 2013, 2015 and 2017. They can all coexist side by side and you'll be able to test them during development. For Visual Studio 2017, even different editions such as Community, Professional and Enterprise can coexist at the same time, something that wasn't possible with older versions of Visual Studio. If available space is a concern, install the minimal components for the old versions, or skip some version in the middle of the range (2013 or 2015).

On your second development machine, install only Visual Studio 2017 or, even better, a build server with no Visual Studio version installed (just the Build Tools 2017), to build your extension for release. This approach will help ensure that you're not inadvertently using DLLs or other dependencies from folders installed by older Visual Studio versions. You might also wonder if it wouldn't be safer to develop or build on a machine with only Visual Studio 2012 installed and the answer is that it's not possible: To generate a VSIX file for Visual Studio 2017 (which creates a version 3 manifest and adds the `catalog.json` and `manifest.json` files), you need the Visual Studio SDK 15.0 of Visual Studio 2017 or, with some work, the Visual Studio SDK 14.0 of Visual Studio 2015. Neither the Visual Studio SDK 12.0 of Visual Studio 2013 nor the Visual Studio SDK 11.0 of Visual Studio 2012 can generate VSIX files for Visual Studio 2017.

And the best practice for (serious) testing is: Use a separate machine (virtual or cloud-based) for each Visual Studio version (so you'll need four machines to test your extension on Visual Studio 2012 to Visual Studio 2017 in isolation). This best practice helped me to find some errors in the code sample for this article!

And the best practice for (serious) testing is: Use a separate machine (virtual or cloud-based) for each Visual Studio version.

To get the Visual Studio 2017 project templates to create a package (or any other kind of extension) you need the "Visual Studio extension development" workload. If you didn't install it when you first installed Visual Studio 2017, go to the folder `C:\Program Files (x86)\Microsoft Visual Studio\Installer`, launch `vs_Installer.exe`, click the Modify button and select that workload at the bottom of the list.

Create a new VSIX project using the File | New | Project menu; go to the Visual C# | Extensibility templates; ensure you've selected .NET Framework 4.5 on the dropdown list at the top; and select the



DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.



Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

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

VSIX Project template. Name the project VSIX-ProjectVS2012_2017. Double-click the source.extension.vsixmanifest file to open its custom editor. In the Metadata tab, set the product name, author, version and so on. In the Install Targets tab, click the Edit button, select the Microsoft.VisualStudio.Pro identifier (that value also targets the Community edition, which is basically a Professional edition) and set the target installation range, [11.0,15.0], as shown in **Figure 5**. A square bracket means the value is included. A parenthesis would mean that the value is excluded, so you can also set [11.0,16.0). You can also target a minor version (like 15.3) using the build number (such as 15.0.26208.1).

In the Dependencies tab, delete all items. In the Prerequisites tab, click the Edit button and set the minimal Visual Studio 2017 component your extension requires. In this example, only the Visual Studio core editor is required. This section is new for Visual Studio 2017 and the version 3 manifest, so it only applies to version 15.0 (see **Figure 6**):

Add a package to the VSIX project by right-clicking the VSIX project node in Solution Explorer, then select the Add | New Item menu to bring up the Add New Item dialog. Now, go to the Visual Studio C# Items | Extensibility | VSPackage node, select the Visual Studio Package template and name it MyPackage.cs. Add a command to the package repeating the actions of the previous step, but selecting this time the Custom Command template. Name this MyCommand1.cs.

To follow the principle of using the fewest dependencies required, in the source code of MyPackage.cs and MyCommand1.cs, remove the unused (grayed) namespaces. Then right-click the VSIX project node in Solution Explorer and click the Manage NuGet Packages for Solution entry. In the Installed section, uninstall all the packages in the order shown here:

- Microsoft.VisualStudio.Shell.15.0
- Microsoft.VisualStudio.Shell.Framework
- Microsoft.VisualStudio.CoreUtility
- Microsoft.VisualStudio.Imaging
- Microsoft.VisualStudio.Shell.Interop.12.0
- Microsoft.VisualStudio.Shell.Interop.11.0
- Microsoft.VisualStudio.Shell.Interop.10.0
- Microsoft.VisualStudio.Threading
- Microsoft.VisualStudio.Shell.Interop.9.0
- Microsoft.VisualStudio.Shell.Interop.8.0
- Microsoft.VisualStudio.TextManager.Interop.8.0
- Microsoft.VisualStudio.Shell.Interop
- Microsoft.VisualStudio.TextManager.Interop
- Microsoft.VisualStudio.Validation
- Microsoft.VisualStudio.Utilities
- Microsoft.VisualStudio.OLE.Interop

(Don't uninstall the Microsoft.VSSDK.BuildTools package, which is the Visual Studio SDK.)

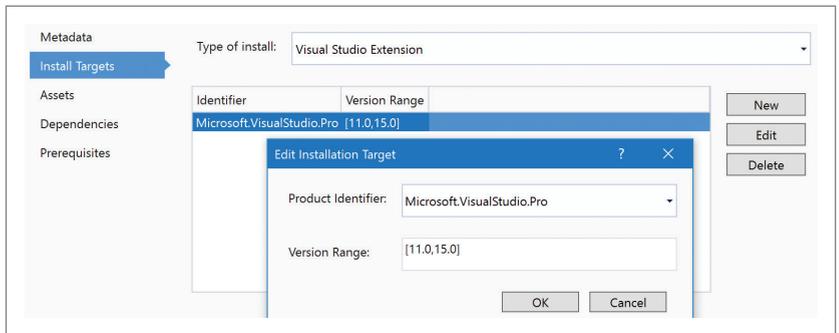


Figure 5 Installation Targets

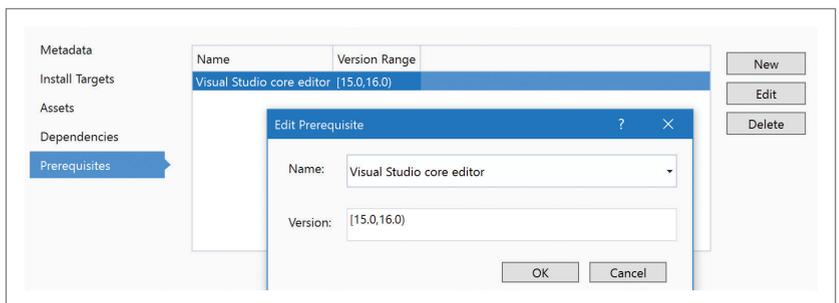


Figure 6 Prerequisites

In the project's References node in Solution Explorer, uninstall all the remaining references (that weren't acquired as NuGet packages) except System and System.Design. Now you can rebuild the solution. You'll get compilation errors that will be solved adding just the references shown in **Figure 7**.

Unfortunately, Microsoft doesn't provide an official NuGet package for Microsoft.VisualStudio.Shell.11.0 (you can find an unofficial NuGet VSSDK.Shell.11 package, though). If you have Visual Studio 2012 installed (you should if that's the minimal-supported version for your extension), you can get it from the GAC as explained earlier. Alternatively, you can get all the required assemblies by installing the Visual Studio 2012 SDK (bit.ly/2mGsfq) that provides them in the subfolders v2.0 and v4.0 of the folder C:\Program Files (x86)\Microsoft Visual Studio 11.0\VSSDK\VisualStudioIntegration\Common\Assemblies. The last column of the table shows the subfolder of the Visual Studio 2012 SDK where you can find each assembly.

To avoid dependencies on unofficial NuGet packages or on specific local folders (either from a Visual Studio SDK or from a

Figure 7 Visual Studio 2012 References

Assembly Name	Assembly Version	Visual Studio 2012 SDK Subfolder
Microsoft.VisualStudio.OLE.Interop	7.1.40304.0	v2.0
Microsoft.VisualStudio.Shell.Interop	7.1.40304.0	v2.0
Microsoft.VisualStudio.Shell.Interop.8.0	8.0.0.0	v2.0
Microsoft.VisualStudio.Shell.Interop.9.0	9.0.0.0	v2.0
Microsoft.VisualStudio.Shell.Interop.10.0	10.0.0.0	v2.0
Microsoft.VisualStudio.Shell.Immutable.10.0	10.0.0.0	v4.0
Microsoft.VisualStudio.Shell.11.0	11.0.0.0	v4.0

Instantly Search Terabytes of Text

Executive Summary

- The dtSearch enterprise and developer product line instantly searches terabytes of text, with no limit on the number of concurrent search threads.
- dtSearch's own document filters support a wide variety of data formats, including "Office" files, PDFs, emails and attachments, online data and other databases.
- The products offer over 25 hit-highlighted search options, with special forensics search options and extensive international language support.
- Developer products include faceted searching and multiple other advanced data classification options.
- SDKs span a wide range of platforms, with APIs for .NET, C++ and Java.

Key Benefits

Terabyte Indexer. dtSearch enterprise and developer products can index over a terabyte of text in a single index, spanning multiple directories, emails and attachments, online data and other databases. dtSearch products can create and search any number of indexes, and can search indexes during updates.

Concurrent, Multithreaded Searching. dtSearch developer products support efficient multithreaded searching, with no limit on the number of concurrent search threads.

Document Filters and Supported Data Types. dtSearch's own document filters support "Office" documents, PDFs, compression formats, emails and multilevel nested attachments, online data and other databases. Document filters are built into the product line and also available for separate licensing.

Over 25 Search Options. dtSearch products have more than 25 search features, including special forensics search options and extensive international language support. The dtSearch Engine also has a range of relevancy-ranking options, including positive

The infographic features the dtSearch logo at the top left, which includes a magnifying glass over a circuit board. The background is a blue and white geometric pattern. Five white diamond-shaped callouts contain the following text:

- The Smart Choice for Text Retrieval® since 1991, dtSearch has resellers on six continents**
- dtSearch enterprise and developer use spans a wide variety of market segments**
- For example, 3 out of 4 of the "Big 4" accounting firms have dtSearch multiuser or developer licenses**
- And 4 out of 5 of Fortune's largest Aerospace and Defense industry companies are developer customers**
- See dtSearch.com for hundreds of developer case studies and reviews, and fully-functional evaluations**

and negative variable term weighting and metadata ranking. For federated searching, products have integrated relevancy ranking with **multicolor hit-highlighting search options** across both online and offline data.

Faceted Search and Other Search Results Filtering. The dtSearch Engine supports user-interface-driven faceted or "drill down" category searching, as well as numerous other full-text and metadata classification options.

SDKs. The dtSearch Engine offers .NET, Java and C++ APIs. dtSearch.com has extensive code samples covering topics such as faceted searching and indexing databases, including SharePoint, NoSQL and SQL, along with BLOB data. Platforms include Windows and Linux (with separate native 64-bit builds of both) as well as UWP, Mac and Android. The dtSearch Engine also works on cloud platforms like Azure and AWS.

For hundreds of developer case studies and press review, and fully-functional evaluations (including the search engine and the document filters), visit



dtSearch.com

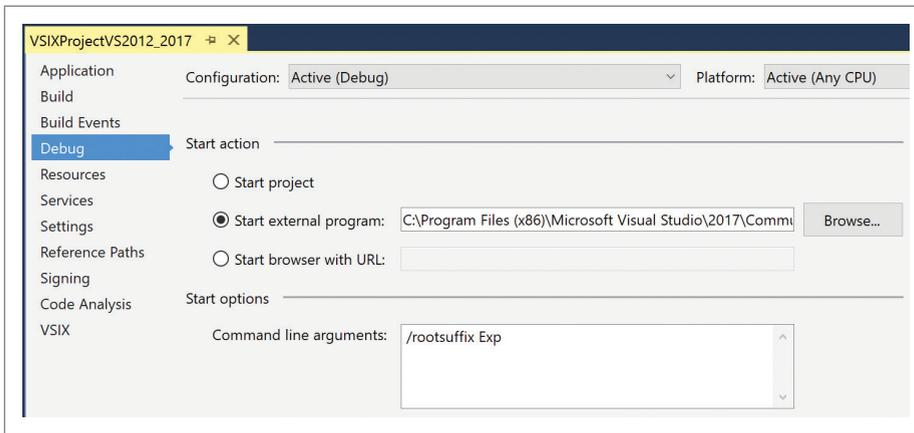


Figure 8 Debug Experimental Instance

Visual Studio installation), the best approach is to get the assemblies from wherever and create a folder called VS2012Assemblies under the root folder of the project. Then, copy the DLLs to that folder, reference them from there (using the Browse button of the project's Reference Manager dialog) and add the VS2012Assemblies folder to source code control, ensuring that the DLLs are added to it (normally source code control tools don't add DLLs by default). So, from this point, the required Visual Studio assemblies are part of the source code.

As you probably know, many Visual Studio projects support “round-tripping”; that is, they can be opened and debugged by several Visual Studio versions without suffering modifications.

To follow the principle of not including assembly references in the VSIX file and not even in the output folder, select each reference and in the Properties window ensure that the Copy Local property is set to False. At this point the solution can be rebuilt without errors. Using Windows Explorer, go to the output folder. Only these files should be generated: extension.vsixmanifest, VSIXProjectVS2012_2017.dll, VSIXProjectVS2012_2017.pkgdef and VSIXProjectVS2012_2017.vsix.

When you build the project, one of the MSBuild targets deploys the extension to the Experimental instance of Visual Studio. This is an instance of Visual Studio that uses different folders and Registry entries than the normal instance, so that you don't make the normal instance unusable if something goes wrong with your extension during development. (You can always reset the Experimental instance clicking the Windows Start button, typing

“Reset the” and executing the “Reset the Visual Studio 2017 Experimental Instance” command.) If you go to the Debug tab on the Properties page of the project, you can set the Start external program field to the Visual Studio 2017 devenv.exe file. (It's important to change this if upgrading, since it would point to an old version of Visual Studio.) You can also see that the Command line arguments specify “Exp” as the root suffix (see Figure 8), so that the Experimental Instance is also used for debugging.

Click the Debug | Start Debugging menu entry and a new Visual Studio

instance will be launched (notice its caption indicates “Experimental Instance”). If you click the Tools | Invoke MyCommand1 menu entry, the package will be loaded, the command will be executed and a message box will be shown.

If you want to use Visual Studio 2017 to debug the extension on a previous Visual Studio version, you need to make two changes: First, because once the extension is built it's deployed to the Visual Studio Experimental Instance of the version whose SDK was used to build the project, you need to remove the NuGet package Microsoft.VSSDK.BuildTools version 15.0 and use version 14.0 for Visual Studio 2015 or version 12.0 for Visual Studio 2013. For Visual Studio 2012 there isn't a NuGet package for the VSDK, so you need to edit the .csproj file and point the VSToolsPath variable to the location of the VSSDK 11.0 (C:\Program Files (x86)\MSBuild\Microsoft\VisualStudio\v11.0), which you must install separately. Second, you need to go to the Debug tab on the Properties page of the project and set the Start external program field to the matching Common7\IDE\devenv.exe executable.

As you probably know, many Visual Studio projects support round-tripping. That is, they can be opened and debugged by several Visual Studio versions without suffering modifications. This is not the case with extensibility projects “out of the box.” However, with some mastering of MSBuild and Visual Studio SDKs, you may achieve it, but it's always a tricky approach.

Once you're done with the development and debugging, you can build your extension in Release configuration and test it on Visual Studio versions installed in isolated instances on test machines. If everything goes well, you can then publish your extension on the Visual Studio Marketplace! ■

CARLOS QUINTERO has received the Microsoft Most Valuable Professional award 14 times, currently in the category of Visual Studio and Development Technologies. He has been helping other developers to create extensions for Visual Studio since 2002, blogging about it since 2006 at visualstudioextensibility.com and more recently tweeting about it: @VSExtensibility.

THANKS to the following technical experts for reviewing this article:
Justin Clareburt, Alex Eyler and Mads Kristensen

File Format APIs

Working with Files?

CREATE CONVERT PRINT
MODIFY COMBINE

FREE TRIAL



Aspose.Total

Manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats in your applications without installing Microsoft Office.

DOC, XLS, PDF, PPT, MSG, BMP, PNG, XML and many more!

Platforms supported: .NET, Java, Cloud, Android, SharePoint, Reporting Services, and JasperReports

CONTACT US

US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987

sales@asposeptyltd.com

Try for FREE at
www.aspose.com

 **ASPOSE**
File Format APIs

How Xamarin.Forms Customization Took an FAA Drone App Higher

Dan Hermes

More than 1 million drones are in the hands of recreational flyers. People are taking unprecedented video footage of events, geography and nature with drone cameras. Commercial drone flyers are conducting inspections of structures and surveying land in a way that's changing their industries. All these drones in the air have become the concern of the Federal Aviation Administration (FAA), which has responded with a strategy and a series of new regulations aimed at helping flyers operate safely and legally.

Of course, there's an app for that. It's called B4UFLY, and it's written in Xamarin.Forms (*note: B4UFLY is used with permission from Network Designs and the FAA*). Drawing upon FAA airport and special location data, the app provides flyers with an interactive map and real-time status updates depending on their position or their planned flight. The status reflects the level of flight safety and legality and helps the flyer find areas away from airports and restricted airspace. The app, as shown in **Figure 1**, has been downloaded more than 300,000 times and is in its second year of updates.

This article discusses:

- Xamarin.Forms customization
- Cross-platform mobile development
- Dynamic layouts

Technologies discussed:

Xamarin.Forms, Custom Renderer, Effects, Native View Declaration

The beauty of this Xamarin.Forms implementation is just how much of it is truly cross-platform. Of the 25 screens in the app, only one requires platform-specific customization. Many, if not most, mobile app requirements today include a cross-platform mandate. If such an app plan is mostly data entry and display, standard navigation and UI, and minimal graphics and animation, then it should be considered a strong candidate for development using Xamarin.Forms.

What Is Xamarin.Forms?

Xamarin.Forms is a library of cross-platform UI classes built atop Xamarin.Android and Xamarin.iOS that also binds directly to the native Universal Windows Platform (UWP), as shown in **Figure 2**. This provides a cross-platform set of UI components that render in each of the three native OSes.

Xamarin.Forms provides a cross-platform library of pages, layouts, and controls and is a great place to begin building an app quickly. There are two ways to create UIs in Xamarin.Forms: either in C# using the rich Xamarin.Forms API or using Extensible Markup Language (XAML), a declarative markup language created by Microsoft.

What Does the Xamarin.Forms Solution Look Like?

The B4UFLY solution contains four projects. The B4UFLY project contains the Xamarin.Forms markup and code. The b4ufly.Droid project contains the Android-specific code and the b4ufly.iOS

project is the iOS piece of the solution. B4UFLy_UITEST contains scripts for UI testing, which can be done on a local computer or, ultimately, on Xamarin Test Cloud.

The Xamarin.Forms project, called B4UFLy, contains cross-platform UI code written using XAML with C# codebehind and the Xamarin.Forms library. Cross-platform business logic and data access code is housed in the UTILS folder. App.cs is the initialization file for the Xamarin.Forms app.

Each platform-specific project has its own startup file for the respective OS. The Android project contains a startup file called MainActivity.cs, which defines an activity class inherited from Xamarin.Forms.Platform.Android.FormsApplicationActivity.

The iOS project contains a startup file called AppDelegate, which inherits from Xamarin.Forms.Platform.iOS.FormsApplicationDelegate.

Once a Xamarin.Forms project is created, development of the UI can follow.

Dynamic Layouts

B4UFLY makes use of all of the standard Xamarin layouts, including StackLayout, AbsoluteLayout and Grid. Xamarin.Forms Layouts can also be employed to create dynamic layouts with content that changes in real time. This isn't about data binding, although that's possible, as well. This is about modifying the structure and appearance of the screens themselves.

The two most important screens in the app are the map and the status page. The map is where the flyer's GPS position is determined, and where surrounding locations and flight restrictions and airports are displayed. The map is also where a pin can be dropped, in something called Planning Mode, so the flyer can determine if it's safe to fly there.

The status page (Figure 3) tells the user if it's safe to fly. There are three main statuses: yellow, orange and red. (There's no green because of lawyers.) Each of these statuses is reflected on the status page by a different status icon, by the text in the header and the color of the header's background, as well as the text that's displayed on the page to explain the status. Even the additional info buttons at the bottom of the page can change. The entire status page is dynamic.

Xamarin.Forms provides several ways to change content in midstream, providing dynamic content modifiable in real time. The first way is to modify existing layouts and their elements. The second is to show and hide elements. The third way is to add and remove elements from the page using C#. B4UFLY employs all three of these approaches in the status screen.

Modifying layouts begins with a layout to modify, created using XAML in this case, though it could just as easily be created using C#. This example is a StackLayout containing a status bar at the top of the map containing a status icon called topStatusIcon:

```
<StackLayout x:Name="topStatusIconHolder" Orientation="Horizontal"
VerticalOptions="FillAndExpand" HorizontalOptions="StartAndExpand"
Padding="0, 5, 5, 0" BackgroundColor="White" >
<Image x:Name="topStatusIcon" Aspect="AspectFit" Source="Blank.png"
VerticalOptions="CenterAndExpand"
BackgroundColor="Transparent" HorizontalOptions="CenterAndExpand"
HeightRequest="50" WidthRequest="50" />
</StackLayout>
```

Depending on the user's flight location, the status can change to fly or no-fly. This example shows a no-fly situation and the text and icon are updated to reflect the restriction:

```
if (safeToFlyResult.isInForbiddenZone == true)
{
topStatusTextHolder.BackgroundColor = Color.White;
topStatusText.Text = "Flight Prohibited";
topStatusText.IsVisible = true;
topStatusIcon.Source = ImageSource.FromFile("no_drone_zone.png");
}
```

Showing and hiding elements begins with a XAML layout, the "DO NOT FLY" status in this case:

```
<StackLayout x:Name="stackForbiddenToFly" Orientation="Vertical" IsVisible="false"
Padding="10, 20, 10, 5" VerticalOptions="Start">
<Label x:Name="forbiddenDoNotFlyText" Text="DO NOT FLY YOUR AIRCRAFT"
TextColor="#DA4E5B"
FontSize="22" FontAttributes="Bold" HorizontalOptions="Center"
HorizontalTextAlignment="Center" />
</StackLayout>
```

When the status is determined to be no-fly because a location has been chosen where drone flight is prohibited, the StackLayout stackForbiddenToFly is made visible (as shown in Figure 3):

```
if (safeToFlyResult.isInForbiddenZone == true)
{
stackForbiddenToFly.IsVisible = true;
...
}
```

The final dynamic UI approach is the physical removal of elements from a layout using C# code. Here's an example of a layout and button being removed from a layout's collection of child elements:

```
stackCurrentLocationTop.Children.Remove (refreshComboStack);
stackCurrentLocationTop.Children.Remove (dismissImgBtn);
```

Add a layout to a layout's children:

```
stackCurrentLocationTop.Children.Add
(refreshComboStack, 3, 4, 0, 1);
```

Those are the three main approaches to dynamic UI: modify existing layouts and their elements, showing and hiding elements, and adding and removing elements and layouts from layout collections using C#.

Xamarin.Forms has become an increasingly easier choice with the outstanding support for Xamarin.Forms customization, providing access to native UI features. A good rule of thumb is that you don't want to have to customize (by platform) more than 20 percent to 30 percent of your app. More than that and you should use a platform-specific option, such as Xamarin.Android or Xamarin.iOS. So what does it mean to customize a Xamarin.Forms app?

Customizing Your App Using Xamarin.Forms

Before Xamarin.Forms was released, I would code my mobile app's cross-platform business logic and data layer in C#. I

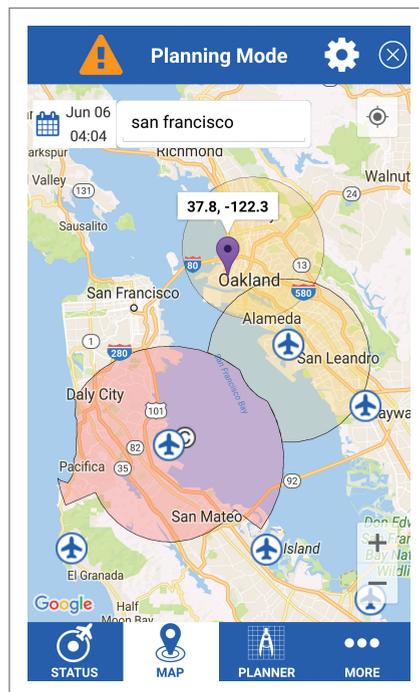


Figure 1 B4UFLY Planning Mode Helps People Find Places to Fly Their Drones

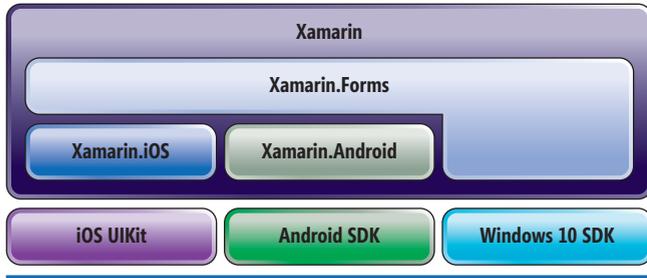


Figure 2 Xamarin Libraries Bind to Native OS Libraries

would then build my UIs with complete access to the underlying native SDKs, but I'd need to make UIs for each platform using Xamarin.iOS, Xamarin.Android or Windows 10 SDK.

So when it was first announced that with Xamarin.Forms you could build your mobile UI only once and compile for iOS, Android and the UWP, my heart skipped a beat. That's because it's what I always longed for: an end-to-end cross-platform development experience.

However, I knew just how deep Xamarin already went when it came to native UI and I wondered: "What if I need something that Xamarin.Forms can't do?"

I asked everyone I knew to explain exactly what Xamarin.Forms could do and what it couldn't do, and I received many terrific responses that helped me better understand Xamarin.Forms, but no one could really answer my question. So, I wrote a book to answer it: "Xamarin Mobile Application Development" (Apress, 2015). And here's a spoiler: Use custom renderers.

Custom renderers give you the ability to punch down through the Xamarin.Forms abstraction and gain direct access to Xamarin.Android, Xamarin.iOS and the UWP. This means access

to the native UI SDKs: iOS UIKit, Android SDK and Windows 10 SDK. You can create platform-specific views and pages in the platform-specific project anytime you need to use functionality in native iOS, Android and Windows.

Using the Xamarin.Forms built-in Dependency Injection, you initialize and reference the custom UI class and Xamarin pulls it out of the appropriate platform's project for you. That's how the map page was built in B4UFLY.

But what if you just want to change one or two properties or events and don't need an entire custom UI class?

Enter Effects. Coding an entire UI renderer class for each platform can be excessive. Sometimes all that's needed is a tweak to a single control element, such as a drop shadow on a label. While custom renderers expose an entire platform-specific class, Effects exposes just its properties. The entire element needn't be subclassed, though a platform-specific class is necessary. A Xamarin.Forms effect offers this precision approach to platform-specific UI customization.

What if all you really need is a native control on your Xamarin.Forms layout?

Take the plunge and declare a platform-specific control, sometimes called a "native control," though it's a Xamarin control and not truly native. Instead of coding overly customized custom renderers, declare native views from Xamarin.iOS, Xamarin.Android or the UWP directly into your Xamarin.Forms layouts. Using a shared project and conditional compilation, include platform-specific UI libraries in your C# UI classes where you can reference them as directly as if you were coding in the native platform. Set properties and event handlers on these views and use them side-by-side with Xamarin.Forms views, in both C# and XAML.

Xamarin.Forms development gives you the ease of cross-platform development using C# and a single UI library with a solid

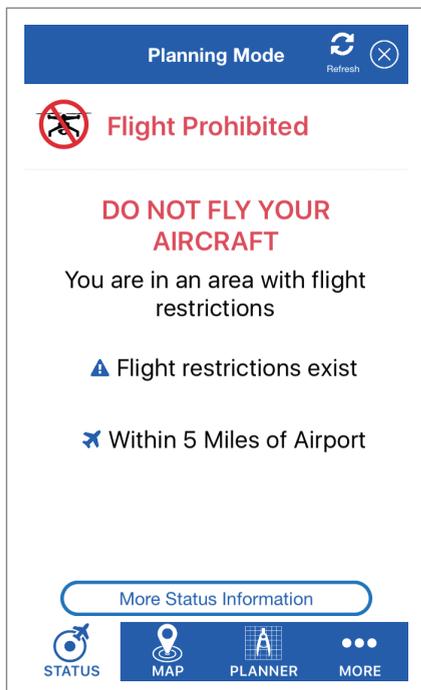


Figure 3 B4UFLY Status Page in a No-Fly Area

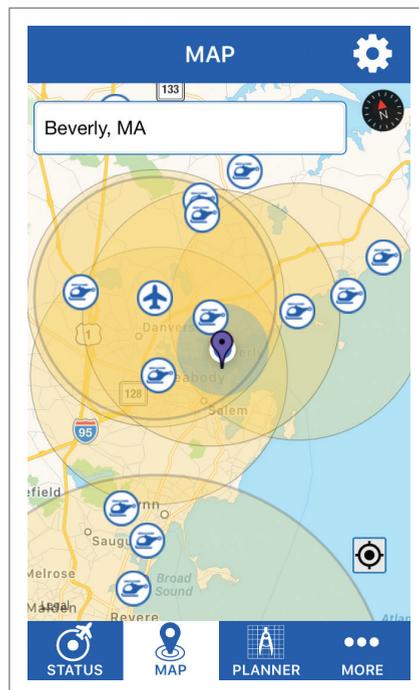


Figure 4 Map Page at Lexicon Systems Office in Beverly, Mass.

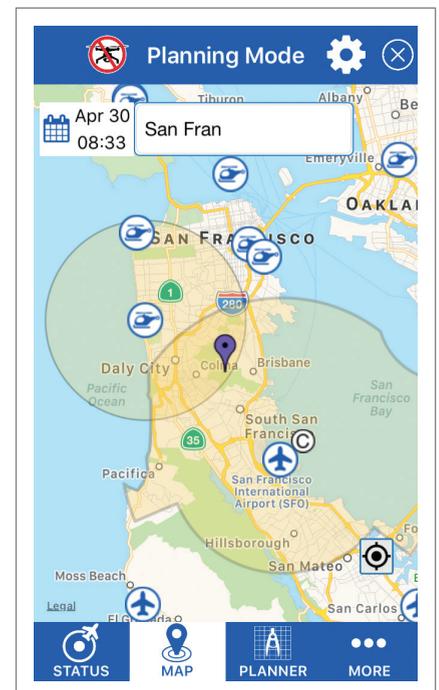


Figure 5 B4UFLY Planning Mode Page in San Francisco, Calif.



JetBrains Rider: New Cross-Platform .NET IDE

A Q&A with Kirill Skrygan,
Rider Team Lead at JetBrains

Q What is JetBrains Rider?

A Rider is our new stand-alone cross-platform IDE for .NET development. It is built on top of IntelliJ platform and incorporates features of our well-known Visual Studio extension, ReSharper. Rider is a smart, powerful, yet very fast and smooth IDE.

Rider provides .NET developers with smart code completion, highlighting, search, navigation, code inspections, quick-fixes, and refactorings. IntelliJ platform brings debugger, excellent VCS integration, local history, building, and many other features that help developers be productive.

We are doing our best to make Rider fast, because it's one of the key aspects that affect developer happiness. Rider uses our new technology that runs features like code indexing or analysis in a process that is completely separated from UI, which allows Rider to be a very powerful, yet smooth and responsive IDE, that lets you develop .NET applications on Mac or Linux, not just Windows.

Q What platforms and technologies does Rider support?

A The majority of modern .NET technologies is supported, including ASP.NET and ASP.NET Core web applications, as well as desktop .NET applications.

We support Unity via bundled plugin that brings Unity-specific features to Rider. Xamarin is supported on both iOS/Android platforms, so Rider can load, build, run and debug Xamarin applications.

Rider embeds IntelliJ components to better support different .NET-related technologies. For example, it uses amazing features from WebStorm to support all kinds of web technologies, and DataGrip functionality to work with databases and SQL files.

Q Is Rider going to support everything ReSharper supports?

A Right now Rider has 90% of ReSharper features. Diagrams, call tracking and hierarchy tool windows will ultimately be supported, too. Moreover, both IntelliJ and ReSharper plugins can be used with Rider as well.

Q Why should Visual Studio users consider trying Rider?

A If we're talking about pure Visual Studio experience (without ReSharper), the reasons are fairly obvious: Rider will give you what Visual Studio simply cannot: hundreds of code inspections and fixes, small and large refactorings, smart navigation and code generation.

Compared to Visual Studio with ReSharper installed, Rider still gives you strong reasons to consider it. It's faster (according to feedback we receive from our users) and because it uses 64-bit architecture, Rider can work with very complex or large solutions that are beyond 32-bit capabilities of current Visual Studio. And don't forget that Rider can work on any modern OS, not just Windows.

```
34 </div>
35 </div>
36 </div>
37 <div class="form-group">
38 <div class="col-md-offset-2 col-md-10">
39 <input type="submit" value="Log in" class="btn btn-default" />
40 </div>
41 </div>
42 <p>
43 <a asp-action="Register">Register as a new user?</a>
44 </p>
45 <p>
46 <a asp-action="
47 </fo
48 </section
49 </div>
50 <div class="
51 <section
52 <@awa
53 </section
54 </div>
55 </div>
56 </div>
57
```

To learn more about Rider and download
a free 30-day trial, please visit

→ www.jetbrains.com/rider

foundation of customization options for those times when you really need native features. Use custom renderers to build platform-specific UI classes using Xamarin.iOS, Xamarin.Android and the UWP. Use effects to access platform-specific properties. And when you can't do without the real thing, declare a native view in your Xamarin.Forms layout.

Custom Renderers—B4UFLY Map

The B4UFLY map page is the only page of more than 25 in the app that requires customization. That ratio of 25:1 generic Xamarin.Forms pages to customized page makes this app a strong case study for Xamarin.Forms.

The map uses your current location and provides immediate surrounding flight restrictions and warnings, as shown in **Figure 4**.

A variation on the map page is Planning Mode, which permits the dropping of a pin to determine the restrictions and flight statuses of hypothetical locations, as shown in **Figure 5**. Note the icon in the upper left indicating “no-fly” due to a nearby controlled airspace (the “C” icon).

Xamarin.Forms binds to only a fraction of the features available in the complete platform-specific UI libraries (iOS Webkit, Android SDK and Windows 10 SDK). Fortunately, Xamarin.Forms exposes the mechanism whereby cross-platform views are converted into platform-specific views. This mechanism is called rendering. By creating your own custom renderers, you get full access to platform-specific features buried deep within each view.

Custom renderers are a bridge between Xamarin.Forms and Xamarin platform-specific libraries, Xamarin.iOS, Xamarin.Android and Windows 10 SDK. Think of a custom renderer as a way to access and extend the binding between Xamarin.Forms and the platform-specific elements.

Project requirements call for features not possible with the out-of-the-box Xamarin.Forms.Maps library, including the placement of icons and colored areas around each icon to delimit certain airspaces on the map. Custom rendering to the rescue! Beginning with `MapPage`, created by inheriting `ContentPage`, you can create a foundational class, which you can use to customize its renderer for each platform, letting you code custom graphics separately for iOS and Android:

```
namespace b4ufly.iOS
{
    public partial class MapPage : ContentPage
    {
        public static MapPage me = null;
        public static MyMap map = null;
        public static Boolean plannerModeOn = false;
    }
}
```

Once you have a custom element, `MapPage`, then you need to create the custom renderers for each platform, iOS and Android in B4UFLY, although you can also do this for UWP. Renderers realize a view on the native platform. You create your own renderer by inheriting from the standard `MapRenderer`, beginning with iOS:

```
[assembly:ExportRenderer (typeof(MyMap), typeof(MyMapRenderer))]
namespace b4ufly.iOS
{
    public class MyMapRenderer : MapRenderer, MapExtension
    {
    }
```

`MyMapRenderer` draws the locations on the map that drone flyers need to be aware of: airports, controlled airspace, military

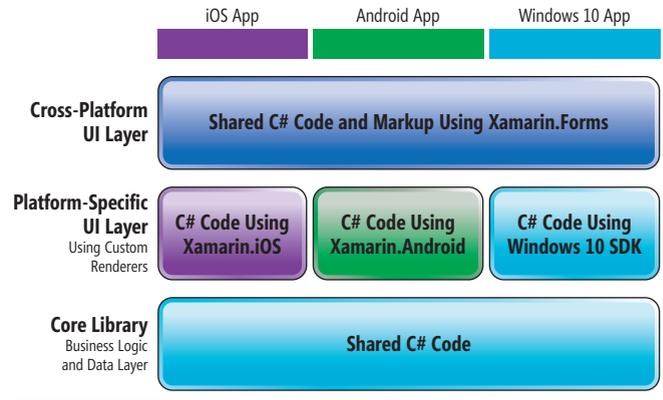


Figure 6 Xamarin.Forms UI

facilities and the like. The renderer draws both icons and surrounding colored areas denoting the important airspace. These types of graphics are handled slightly differently in iOS than in Android. The Android map renderer uses a similar approach to the one used for iOS:

```
[assembly:ExportRenderer (typeof(MyMap), typeof(MyMapRenderer))]
namespace b4ufly.Droid
{
    public class MyMapRenderer : MapRenderer, MapExtension,
        GoogleMap.IOnCameraChangeListener, GoogleMap.IOnMarkerDragListener,
        GoogleMap.IOnMarkerClickListener
    {
    }
```

Once you create the renderers, it's time to use them. Based on the `MyMap` data type, which uses the `MyMapRenderer`, the following statement instantiates a platform-specific map:

```
map = new MyMap(MapSpan.FromCenterAndRadius(new Position(0, 0), Distance.FromMiles(1.0)))
```

The built-in Inversion of Control (IoC) mechanism in Xamarin.Forms uses the renderer from the platform project currently being built. By adding platform-specific map references, you could explicitly instantiate an Apple Mapkit in the iOS renderer and a Google Map in the Android renderer.

Figure 7 iOS implementation of `DropShadowEffectLabel`

```
[assembly:ResolutionGroupName ("FAA")]
[assembly:ExportEffect (typeof(DropShadowEffectLabel), "DropShadowEffectLabel")]
namespace b4ufly.iOS
{
    public class DropShadowEffectLabel : PlatformEffect
    {
        protected override void OnAttached ()
        {
            try {
                var effect =
                    (DropShadowEffect)Element.Effects.FirstOrDefault(
                        e => e is DropShadowEffect);
                if (effect != null) {
                    Control.Layer.ShadowColor = effect.Color.ToCGColor();
                    Control.Layer.CornerRadius = 5;
                    Control.Layer.ShadowOffset = new CGSize (5, 5);
                    Control.Layer.ShadowOpacity = 1.0f;
                }
            } catch (Exception ex)
            {
                Console.WriteLine ("Cannot set effect property. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }
    }
}
```

Data Quality Made Easy. Your Data, Your Way.



Melissa provides the full spectrum of data quality to ensure you have data you can trust.

We profile, standardize, verify, match and enrich global **People Data** – name, address, email, phone, and more.

Our data quality solutions are available on-premises and in the Cloud – fast, easy to use, and powerful developer tools and plugins for the **Microsoft® Product Ecosystem**.



Start Your Free Trial

www.Melissa.com/msft-pd

Melissa Data is Now Melissa.

Why the change?

See for Yourself at the New www.Melissa.com

melissa[™]

1-800-MELISSA

Customization of `Xamarin.Forms` elements leads you to a different view of the solution architecture, with custom renderers residing in the middle platform-specific UI layer, as shown in **Figure 6**.

Custom renderers are powerful and thorough in their implementation as platform-specific enablers of `Xamarin.Forms` UI elements. Custom renderers are, however, heavy artillery. If you want something more tactical, like merely customizing a property on a `Xamarin.Forms` control, consider an “effect.”

Effects

Effects provide access to individual platform-specific properties of controls and can be parameterized. To create an effect, first create a class that is a subclass of the `RoutingEffect` class. Mind the method overrides and attributes. Then use the effect in your app.

In addition to exposing properties, effects also have the capacity to pass parameters to those properties and define events on `Xamarin.Forms` controls. You pass parameters to the effect using Attached Properties or the Common Language Runtime (CLR). The following example uses the CLR to bind properties to the effect and creates the `DropShadowEffect` in the `Xamarin.Forms` project:

```
public class DropShadowEffect : RoutingEffect
{
    public Color Color { get; set; }

    public DropShadowEffect () : base ("FAA.DropShadowEffectLabel")
    {
    }
}
```

This label effect provides a color property for the shadow and references the platform-specific implementation of the `DropShadowEffectLabel` in its base class.

You implement the effect in a platform-specific project, similarly to a custom renderer, although implementation is optional in each platform. Once per project, you add a `ResolutionGroupName` attribute containing your company name to avoid collisions with other effects of the same name. Each `Effect` class is subclassed

Figure 8 Android Implementation of `DropShadowEffectLabel`

```
[assembly:ResolutionGroupName ("FAA")]
[assembly:ExportEffect (typeof(DropShadowEffectLabel), "DropShadowEffectLabel")]
namespace b4uflly.Droid
{
    public class DropShadowEffectLabel : PlatformEffect
    {
        protected override void OnAttached ()
        {
            try {
                var control = Control as Android.Widget.TextView;
                var effect =
                    (DropShadowEffect)Element.Effects.FirstOrDefault
                    (e => e is DropShadowEffect);
                if (effect != null) {
                    Android.Graphics.Color color = effect.ToAndroid ();
                    control.SetShadowLayer (5, 5, 5, color);
                    // params: radius, offsetX, offsetY, color
                }
            } catch (Exception ex) {
                Console.WriteLine ("Cannot set effect property. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }
    }
}
```

from `PlatformEffect` and needs an `ExportEffect`, which registers the effect with `Xamarin.Forms`. **Figure 7** shows an implementation on iOS in the `Xamarin.iOS` project.

`Control` is an iOS `UIView.PlatformEffect` that exposes these methods and which must be overridden:

- `OnAttached`—customize the control here
- `OnDetached`—perform cleanup (for example, deregister events)

Next is the Android implementation, similar to the iOS effect, except that the label control is the Android-specific `TextView`, as shown in **Figure 8**. The `TextView` control is typed explicitly to access the `SetShadowLayer` method.

Once the effect is in place, it's time to invoke it. First, a control needs to be declared in XAML or C#. You then attach the effect to the control by adding it to the control's effects collection. The following example shows the XAML approach with an `Entry` control declared in XAML with the `DropShadowEffect` added to the control's `Effects` collection and the `Color` property set to black:

```
<Label Text="Label with Shadow" ... >
  <Label.Effects>
    <local:DropShadowEffect Color="Black">
    </local:DropShadowEffect>
  </Label.Effects>
</Label>
```

Using C# instead of XAML, the label with attached effect can be created, as shown here:

```
var label = new Label {
    Text = "Label with Shadow",
    ...
};
label.Effects.Add (new DropShadowEffect {
    Color = Color.Black,
});
```

Tactical customization using effects lets you make specific changes to the `Xamarin.Forms` controls, but sometimes changing certain properties and methods just isn't enough. When you want to use a lot of features of a native control, then you wind up doing a lot of custom effects coding.

Native View Declaration

Sometimes you want complete control of the UI. Thankfully there's now a way to get this in `Xamarin.Forms` via native view declaration. Declared native controls are incredibly powerful, but are not without limitations. They're easiest to use in XAML, secondarily in C# using a `Shared Project` (which is called native embedding), though it's possible but not easy or recommended to use them in a `Portable Class Library (PCL)`. A lot of projects use `PCLs` and that often means native views are best used in XAML, and that's the approach I'll cover here.

There are two steps in declaring a native view in XAML. First, specify the namespace for each native source. Second, declare the native view. **Figure 9** shows an example, using the label control. It begins with the basic XAML page and defines the namespaces for iOS, Android and Windows (shown in bold code).

Next, native views are declared in the `Content` property of the `ContentPage`. A `UILabel` for iOS, a `TextView` for Android and a `TextBlock` for Windows:

```
<ContentPage.Content>
  <ios:UILabel Text="This is an iOS UILabel" View.HorizontalOptions="Start"/>
  <android.widget.TextView Text="This is an Android TextView"
    x:Arguments="{x:Static formsandroid:Forms.Context}" />
  <win:TextBlock Text="This is a Windows TextBlock"/>
</ContentPage.Content>
```

Figure 9 Native Control Namespace Declarations

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;targetPlatform=iOS"
  xmlns:androidWidget="clr-namespace:Android.Widget;assembly=
    Mono.Android;targetPlatform=Android"
  xmlns:formsandroid="clr-namespace:Xamarin.Forms;assembly=
    Xamarin.Forms.Platform.Android;targetPlatform=Android"
  xmlns:win="clr-namespace:Windows.UI.Xaml.Controls;assembly=Windows,
    Version=255.255.255.255, Culture=neutral, PublicKeyToken=null,
    ContentType=WindowsRuntime;targetPlatform=Windows"
  x:Class="b4ufly.NativeView" >
  <ContentPage.Content>
  </ContentPage.Content>
</ContentPage>
```

Those are the three approaches to Xamarin.Forms customization: custom renderers, effects and native view declaration. Custom renderer is a heavyweight option offering a lot of flexibility, while effects provides a surgical approach to customization. Native view declaration is the nuclear option, circumventing Xamarin.Forms entirely.

Wrapping Up

You'll eventually need more from Xamarin.Forms than it gives you out-of-the-box, just like I did with B4UFLY. When complex tasks or designs are required by Xamarin.Forms, virtually anything is possible using Xamarin.Forms customization. Customization provides access to the lower-level, platform-specific, screen-rendering classes called "renderers," which use platform-specific controls to create all Xamarin.Forms screens. Any Xamarin.Forms screen can be broken into platform-specific screens, classes, controls and properties using this approach.

A lighter-weight approach is to use effects to access platform-specific properties and events. You can also use entire native controls on your Xamarin.Forms pages using native view declaration.

This means that you can write a Xamarin.Forms page or app and customize it by platform. Use customization sparingly, or risk a fragmented UI code base that probably should have been written entirely as a platform-specific UI. Used judiciously, customization can turn your basic, lackluster product into a versatile, unique, popular app.

In B4UFLY, the FAA's investment in Xamarin.Forms continues to pay off because of the many ongoing enhancements that are generic to the many cross-platform text-based pages. The platform-specific map page contains some cross-platform elements, but much of that page requires platform-specific customization. This Xamarin.Forms architecture is extensible and development times and costs are lower because of it; the significant code reuse is practical and elegant. ■

DAN HERMES is a Xamarin MVP, a Microsoft MVP, and author of "Xamarin Mobile Application Development." He is principal of Lexicon Systems, a Boston-based consultancy building award-winning mobile apps and helping companies build their own successful apps. Follow his blog at mobilesharpcafe.com, on Twitter: @danhermes or contact him at dan@lexiconsystemsinc.com.

THANKS to the following technical expert for reviewing this article:
Jesse Liberty

msdnmagazine.com



Instantly Search Terabytes of Data

across a desktop, network, Internet or Intranet site with dtSearch enterprise and developer products

Over 25 search features, with **easy multicolor hit-highlighting options**

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

Developers:

- APIs for .NET, Java and C++
- SDKs for Windows, UWP, Linux, Mac and Android
- See dtSearch.com for articles on faceted search, advanced data classification, working with SQL, NoSQL & other DBs, MS Azure, etc.

Visit dtSearch.com for

- hundreds of reviews and case studies
- fully-functional evaluations

The Smart Choice for Text Retrieval®
since 1991

dtSearch.com 1-800-IT-FINDS

Git Internals: Architecture and Index Files

Jonathan Waldman

In my last article (msdn.com/magazine/mt809117), I showed how Git uses a directed acyclic graph (DAG) to organize a repo's commit objects. I also explored the blob, tree and tag objects to which commit objects can refer. I concluded the article with an introduction to branching, including the distinction between HEAD and head. That article is a prerequisite to this one, in which I'll discuss the Git "three-tree" architecture and the importance of its index file. Understanding these additional Git internals will build on the foundational knowledge that will make you a more effective Git user and will provide new insights as you explore various Git operations fronted by the graphical Git tooling in the Visual Studio IDE.

Recall from the last article that Visual Studio communicates with Git using a Git API, and that the Visual Studio IDE Git tooling abstracts away the complexity and capabilities of the underlying Git engine. That's a boon for developers who want to implement a version-control workflow without needing to rely on the Git command-line interface (CLI). Alas, the otherwise helpful Git abstractions of the

IDE can sometimes lead to confusion. For example, ponder the basic workflow of adding a project to Git source control, modifying project files, staging them and then committing the staged files. To do that, you open the Team Explorer Changes pane to view the list of changed files and then you select the ones you want to stage. Consider the leftmost image in **Figure 1**, which shows that I changed two files in the working directory (**Marker 1**).

In the next image to the right, I staged one of those changed files: Program.cs (**Marker 2**). When I did that, Program.cs appears to have "moved" from the Changes list to the Staged Changes list. If I further modify and then save the working directory's copy of Program.cs, it continues to appear in the Staged Changes section (**Marker 3**)—but it also appears in the Changes section (**Marker 4**)! Without understanding what Git is doing behind the scenes, you might be flummoxed until you figured out that two "copies" of Program.cs exist: one in the working folder and one in the Git internal database of objects. Even if you realize that, you might not have any insight as to what would happen when you unstage the staged copy, try to stage the second changed copy of Program.cs, undo changes to the working copy or switch branches.

To truly grasp what Git is doing as you stage, unstage, undo, commit and check out files, you first must understand how Git is architected.

The Git Three-Tree Architecture

Git implements a three-tree architecture (a "tree" in this context refers to a directory structure and files). Working from left to right

This article discusses:

- Git's three-tree architecture
- How the Git index works
- Index extensions

Technologies discussed:

Visual Studio 2017, Git for Windows 2.10

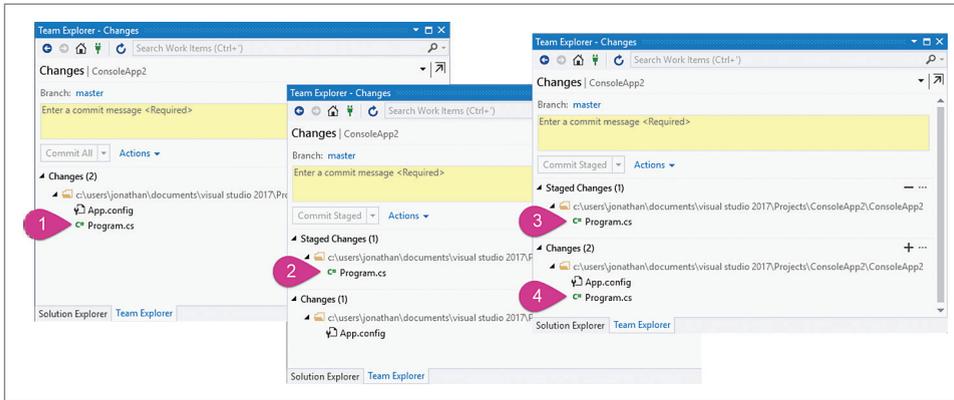


Figure 1 The Team Explorer Changes Pane Can Show the Same File in Its Changes and Staged Changes Sections

in **Figure 2**, the first tree is the collection of files and folders in the working directory—the OS directory that contains the hidden .git folder; the second tree is typically stored in a single binary file called index, located in the root of the .git folder; the third tree is composed of Git objects that represent the DAG (recall that SHA-1-named Git objects are located in two-hex-digit-named folders .git/objects and can also be stored in “pack” files located in .git/objects/pack and in file paths defined by the .git/objects/info\alternates file). Keep in mind that the Git repo is defined by all files that sit in the .git folder. Often, people refer to the DAG as the Git repo, and that’s not quite accurate: The index and the DAG are both contained in the Git repo.

Notice that while each tree stores a directory structure and files, each leverages different data structures in order to retain tree-specific metadata and to optimize storage and retrieval. The first tree (the working directory tree, also called “the working tree”) is plainly the OS files and folders (no special data structures there, other than what’s at the OS level) and serves the needs of the software developer and Visual Studio; the second tree (the Git index) straddles the working directory and the commit objects that form the DAG, thereby helping Git perform speedy working-directory file-content comparisons and quick commits; the third tree (the DAG) makes it possible for Git to track a history of commits, as discussed in the previous article. In its capacity as a robust version control system,

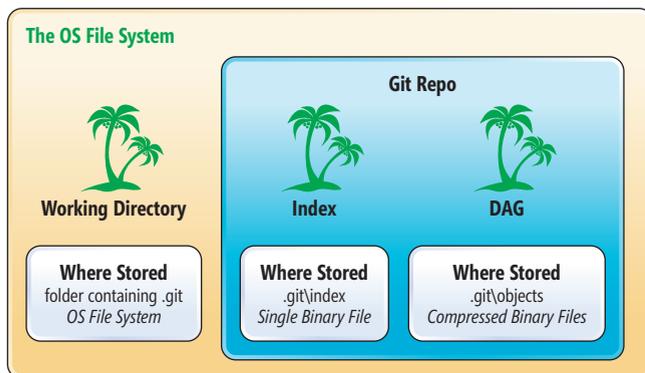


Figure 2 The Git Three-Tree Architecture Leverages the All-Important Index File for Its Smart and Efficient Performance

Git adds helpful metadata to the items it stores in the index and in commit objects. For example, the metadata it stores in the index helps it detect changes to files in the working directory, while the metadata it stores in commit objects helps it track who issued the commit and for what reason.

To review the three trees in the three-tree architecture and to put some perspective around the remainder of this article’s focus: You already know how the working-directory tree functions, because

it’s actually the OS file system you’re already well-versed in using. And if you read my earlier article, you should have good working knowledge of the DAG. Thus, at this point, the missing link is the index tree (hereafter, “the index”) that straddles the working directory and the DAG. In fact, the index plays such an important role that it’s the sole subject of the remainder of this article.

How the Index Works

You might have heard the friendly advice that the index is synonymous with the “staging area.” While that’s somewhat accurate, to speak of it that way belies its true role, which is not only to support a staging area, but also to facilitate the ability of Git to detect changes to files in your working directory; to mediate the branch-merge process, so you can resolve conflicts on a file-by-file basis and safely abort the merge at any time; and to convert staged files and folders into tree objects whose references are written to the next commit object. Git also uses the index to retain information about files in the working tree and about objects retrieved from the DAG—and thus further leveraging the index as a type of cache. Let’s investigate the index more thoroughly.

The index implements its own self-contained file system, giving it the ability to store references to folders and files along with metadata about them. How and when Git updates this index depends on the kind of Git command issued and the command options specified (if you’re so inclined, you can even use the Git update-index plumbing command to manage the index yourself), so exhaustive coverage here isn’t possible. However, as you work with the Visual Studio Git tooling, it’s helpful to be aware of the primary ways in which Git updates the index and in which Git uses information stored in the index. **Figure 3** shows that Git updates the index with working directory data when you stage a file, and it updates the index with DAG data when you initiate a merge (if there are merge conflicts), clone or pull, or switch branches. On the other hand, Git relies on information stored in the index when it updates the DAG after you issue a commit, and when it updates the working directory after you clone or pull, or after you switch branches. Once you realize that Git relies on the index and that the index straddles so many Git operations, you’ll begin to appreciate the advanced Git commands that modify the index, effectively empowering you to finesse how Git operates.

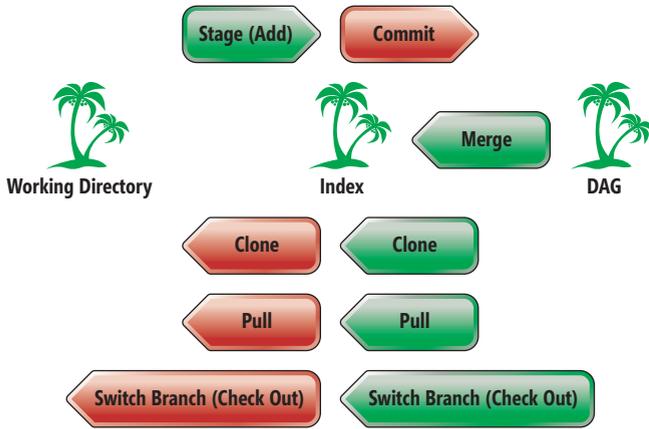


Figure 3 Primary Git Actions That Update the Index (Green) and Git Actions That Rely on What the Index Contains (Red)

Let's create a new file in the working directory to see what happens to it as it's written to the index. As soon as you stage that file, Git creates a header using this string-concatenation formula:

```
blob(space){file-length in bytes}{null-termination character}
```

Git then concatenates the header to the beginning of the file contents. Thus, for a text file containing the string "Hello," the header + file contents would generate a string that looks like this (keep in mind there's a null character before the letter "H"):

```
blob 5Hello
```

To see that more clearly, here's the hexadecimal version of that string:

```
62 6c 6f 62 20 35 00 48 65 6c 6c 6f
```

Git then computes an SHA-1 for the string:

```
5ab2f8a4323abafb10abb68657d9d39f1a775057
```

Git next inspects the existing index to determine if an entry for that folder/file name already exists with the same SHA-1. If so, it

locates the blob object in the .git\objects folder and updates its date-modified time (Git will never overwrite objects that already exist in the repo; it updates the last-modified date so as to delay this newly added object from being considered for garbage collection). Otherwise, it uses the first two characters of the SHA-1 string as the directory name in .git\objects and the remaining 38 characters to name the blob file before zlib-compressing it and writing its contents. In my example, Git would create a folder in .git\objects called 5a and then write the blob object into that folder as a file with the name b2f8a4323abafb10abb68657d9d39f1a775057.

When Git creates a blob object in this manner, you might be surprised that one expected file property is conspicuously missing from the blob object: the file name! That's by design, however. Recall that Git is a content-addressable file system and, as such, it manages SHA-1-named blob objects—not files. Each blob object is normally referenced by at least one tree object, and tree objects in turn are normally referenced by commit objects. Ultimately, Git tree objects express the folder structure of the files you stage. But Git doesn't create those tree objects until you issue a commit. Therefore, you can conclude that if Git uses only the index to prepare a commit object, it also must capture the file-path references for each blob in the index—and that's exactly what it does. In fact, even if two blobs have the same SHA-1 value, as long as each maps to a different file name or different path/file value, each will appear as a separate entry in the index.

Git also saves file metadata with each blob object it writes to the index, such as the file's create and modified dates. Git leverages this information to efficiently detect changes to files in your working directory using file-date comparisons and heuristics rather than brute-force re-computing the SHA-1 values for each file in the working directory. Such a strategy speeds up the information you see in the Team Explorer Changes pane—or when you issue the porcelain Git status command.

Once armed with an index entry for a working-directory file along with its associated metadata, Git is said to "track" the file because it can readily compare its copy of the file with the copy that remains in the working directory. Technically, a tracked file is one that also exists in the working directory and is to be included in the next commit. This is in contrast to untracked files, of which there are two types: files that are in the working directory but not in the index, and files that are explicitly designated as not to be tracked (see the Index Extensions section). To summarize, the index gives Git the power to determine which files are tracked, which are not tracked, and which should not be tracked.

To better understand the specific contents of the index, let's use a concrete example by starting

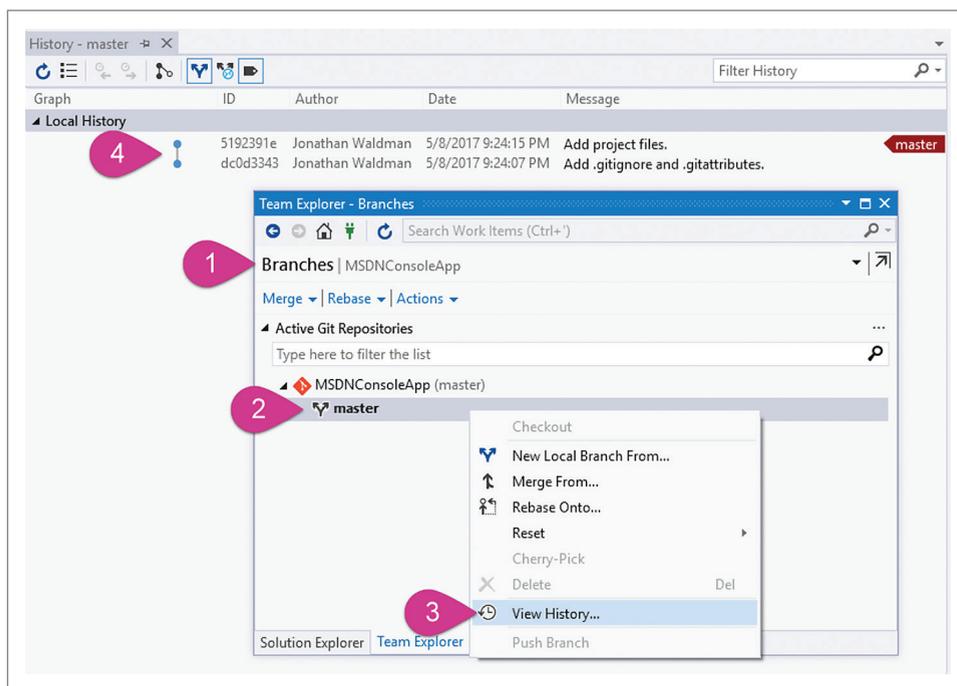


Figure 4 Viewing the History in Order to See What Visual Studio Does When You Create a New Project

Why Choose LEADTOOLS?



OCR	BARCODE
PDF	FORMS RECOGNITION
DICOM/PACS	DOCUMENT VIEWER
MEDICAL VIEWERS	IMAGE PROCESSING
STREAMING	MPEG2-TS

Award-winning Document, Medical, and Multimedia Imaging Technology

LEADTOOLS is a family of comprehensive toolkits designed to help programmers integrate raster, document, medical, multimedia, and vector imaging into their desktop, server, tablet, and mobile applications. LEADTOOLS gives developers the most flexible and powerful imaging technology, offering development support for OCR, Barcode, Forms Recognition, PDF, Document Conversion and Viewing, Document Cleanup, Annotations, DICOM, PACS, HL7, Audio/Video Codecs, MPEG-2 Transport, DVR, Streaming, File Formats (150+), Image Compression, Image Processing, Color Conversion, Viewers, Special Effects, Scanning/Capture, Printing, and more. A LEADTOOLS toolkit literally puts millions of lines of code at the fingertips of application developers.

One-Stop Shop

LEAD supports the full range of imaging categories allowing customers to standardize on LEADTOOLS for all imaging requirements and eliminate multiple-vendor headaches. Enjoy the simplicity of one vendor relationship, one license agreement, and one support contact. With access to the full suite of LEADTOOLS SDKs developers stand to gain Medical Imaging, Document Imaging, Recognition, Vector, Multimedia, and general Imaging components with cross-platform capability.

Free Technical Support

LEAD offers free and unlimited technical support via email, user forums, and live chat; LEADTOOLS customers receive this free technical support in perpetuity. We also offer premium support options on a per case basis.

Imaging Component Experts

LEAD's developers are imaging experts as well as development component experts. Consistently receiving industry recognition from the developer community, LEADTOOLS is designed to allow customers to easily integrate its technology into real-world solutions.

Reduce Time-to-Market

Decrease solution costs and time-to-revenue by leveraging LEADTOOLS imaging libraries and components. Develop your customers' solutions with less overhead and a faster rollout. Deliver sooner and get paid quicker.

Plug In and Go

Quickly image-enable existing solutions with minimal code changes. LEADTOOLS provides many customizable, ready-built components including document and medical web viewers, file converters, recognition engines, and codecs.

Outshine Your Competitors

Increase your company's competitive advantage by adding significant functionality to your offering without substantially affecting cost or time-to-deliver.

Experience that Matters

Founded in 1990, LEAD has a track record of more than 27 years of profitable operations and excellent customer service. LEAD is currently shipping version 19 of its core products. If your customer has an imaging requirement, chances are LEAD has already coded, tested, and deployed it. Increase the productivity of your employees by allowing them to leverage upon LEAD's years of development blood, sweat, and tears.

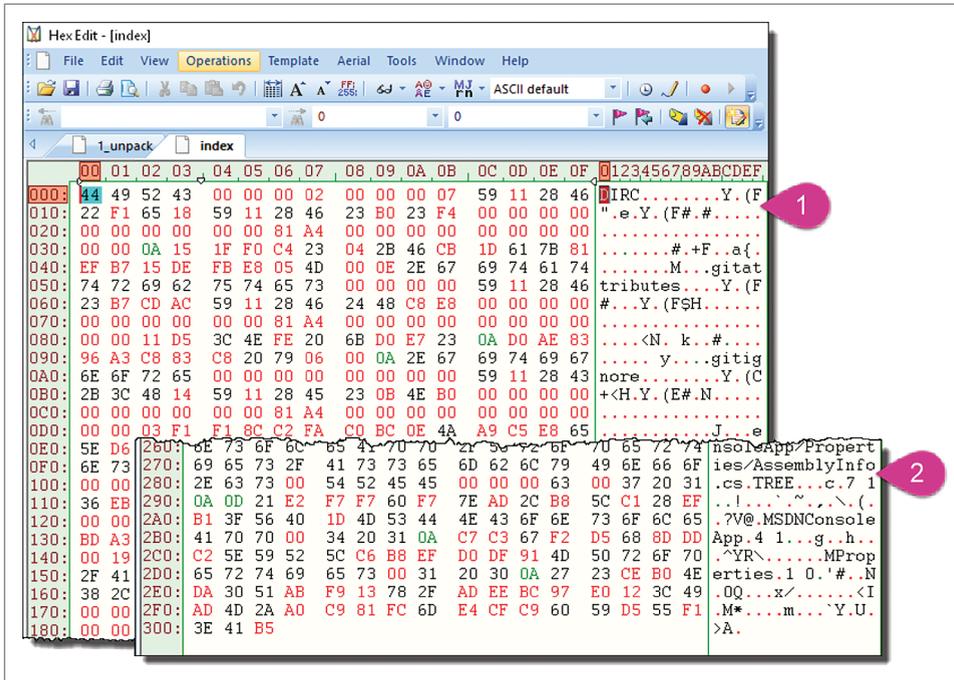


Figure 5 A Hex Dump of the Git Index File for the Project

with a new Visual Studio project. The complexity of this project isn't so important—you just need a couple of files to adequately illustrate what's going on. Create a new console application called `MSDNConsoleApp` and check the `Create directory for solution` and the `Create new Git repository` checkboxes. Click OK to create the solution.

I'll issue some Git commands in a moment, so if you want to run them on your system, open a command prompt window in the working directory and keep that window within reach as you follow along. One way to quickly open a Git command window for a particular Git repo is to access the Visual Studio Team menu and select `Manage Connections`. You'll see a list of local Git repositories, along with the path to that repo's working directory. Right-click the repo name and select `Open Command Prompt` to launch a window into which you can enter Git CLI commands.

Once you create the solution, open the Team Explorer Branches pane (Figure 4, Marker 1) to see that Git created a default branch called `master` (Marker 2). Right-click the `master` branch (Marker 2) and select `View History` (Marker 3) to view the two commits Visual Studio created on your behalf (Marker 4). The first has the commit message "Add .gitignore and .gitattributes"; the second has the commit message "Add project files."

Open the Team Explorer Changes pane. Visual Studio relies on the Git API to populate items in this window—it's the Visual Studio

Figure 6 The Git Index Header Data Format

Index File - Header Entry		
00 - 03 (4 bytes)	DIRC	Fixed header for a directory cache entry. All index files begin with this entry.
04 - 07 (4 bytes)	Version	Index version number (Git for Windows currently uses version 2).
08 - 11 (4 bytes)	Number of entries	As a 4-byte value, the index supports up to 4,294,967,296 entries!

version of the Git status command. Currently, this window indicates there are no unstaged changes in the working directory. The way Git makes this determination is to compare each index entry with each working directory file. With the index's file entries and associated file metadata, Git has all the information it needs to determine whether you've made any changes, additions, deletions, or if you renamed any files in the working directory (excluding any files mentioned in the `.gitignore` file).

So the index plays a key role in making Git smart about differences between your working directory tree and the commit object pointed to by `HEAD`. To learn a bit more about what kind of information the index provides to the Git engine, go to the command-line window

you opened earlier and issue the following plumbing command:

```
git ls-files --stage
```

You can issue this command at any time to generate a complete list of files currently in the index. On my system, this produces the following output:

```
100644 1ff0c423042b46cb1d617b81efb715defbe8054d 0 .gitattributes
100644 3c4efe206bd0e7230ad0ae8396a3c883c8207906 0 .gitignore
100644 f18cc2fac0bc0e4aa9c5e8655ed63fa33563ab1d 0 MSDNConsoleApp.sln
100644 88fa4027bda397de6bf19f0940e5dd6026c877f9 0 MSDNConsoleApp/App.config
100644 d837dc8996b727d6f6d2c4e788dc9857b840148a 0 MSDNConsoleApp/
MSDNConsoleApp.csproj
100644 27e0d58c613432852eab6b9e693d67e5c6d7aba7 0 MSDNConsoleApp/Program.cs
100644 785cfad3244d5e16842f4cf8313c8a75e64adc38 0 MSDNConsoleApp/Properties/
AssemblyInfo.cs
```

The first column of output is a Unix OS file mode, in octal. Git doesn't support the full range of file-mode values, however. You're likely to only ever see 100644 (for non-EXE files) and 100755 (for Unix-based EXE files—Git for Windows also uses 100644 for executable file types). The second column is the SHA-1 value for the file. The third column represents the merge stage value for the file—0 for no conflict or 1, 2 or 3 when a merge conflict exists. Finally, notice that the path and file name for each of the seven blob objects are stored in the index. Git uses the path value when it builds tree objects ahead of the next commit (more on that in a moment).

Now, let's examine the index file itself. Because it's a binary file, I'm going to use HexEdit 4 (a freeware hex editor available at hexedit.com) to view its contents (Figure 5 shows an excerpt).

The first 12 bytes of the index contain the header (see Figure 6). The first 4 bytes will always contain the characters `DIRC` (short for directory cache)—this is one reason the Git index is often referred to as the cache. The next 4 bytes contain the index version number, which defaults to 2 unless you're using certain features of Git (such as sparse checkout), in which case it might be set to version 3 or 4. The final 4 bytes contain the number of file entries contained further down in the index.



Develop Collaborative Web Applications with DocuVieware

Q&A with Loïc Carrère, owner, and CEO of ORPALIS, and creator of GdPicture.NET and DocuVieware

Q You have just released the version 3 of DocuVieware, can you tell us more about this SDK?

A DocuVieware is an HTML5 viewer and document management kit to build web applications. It can be effortlessly integrated into applications based on different web technologies, like angular2, Node.js, php, asp.net mvc/core, etc., with the help of Web Service Architecture.

DocuVieware operates seamlessly on any device and platform, including mobile phones and tablets.

We help developers creating dynamic web applications with user interaction. All User Interface elements and document's appearance in the viewer, like rotation and zooming, are managed client-side. The user can perform a large variety of actions, like creating annotations and comments, and scan and print documents.

Our customers have been using our GdPicture.NET SDK for desktop applications, for many years, and with great success. Some of them have already made the transition to web applications. That's why we have developed a tool which is comprehensive and fully customizable, yet easy to integrate.

DocuVieware is powered by our GdPicture.NET Document Imaging and Image Processing SDK. To build DocuVieware, we have developed a layer containing features specific to web applications, which encapsulates GdPicture.NET and its three thousand functionalities.

Q How do you make DocuVieware evolve?

A DocuVieware is growing every day thanks to customer feedback: if there is something our toolkit doesn't do yet, and you want it included, just ask us, and will see how we can implement it. Customers under maintenance can influence our roadmap significantly to make DocuVieware tailored to their needs.

For version 3, we have tried to focus on making collaborative work an easy task for everyone, thanks to comment statuses and discussion support. Many other features and improvements are also available with this new release.



We're constantly looking for the latest technologies on the market, and we often end up creating our path when we feel something is missing in the industry.

Q DocuVieware is a product of your company, ORPALIS. Do you have any other project coming up?

A We keep on improving our SDK offer by implementing new functionalities in GdPicture.NET and DocuVieware on a weekly basis.

We have a strong expertise in formats: we provide full PDF support and manage more than a hundred formats. We are also specialized in symbol recognition. Our OCR, barcoding, and MICR engines are recognized worldwide for their speed and accuracy.

We are currently working on a new project that combines our technologies and our knowledge in web infrastructures. DocuVieware is already built with a scalable architecture, meaning that your web application developed with DocuVieware will work the same if you're hosting it on one or a thousand servers. Our future innovative platform will offer access to the same components available in GdPicture.NET and DocuVieware (plus a lot of other new innovative features for manipulating documents), in a REST API.

To access your 60 days free trial, go to →

www.docuvieware.com

Following the 12-byte header is a list of *n* index entries, where *n* matches the number of entries described by the index header. The format for each index entry is presented in **Figure 7**. Git sorts index entries in ascending order based on the path/file name field.

The first 8 bytes represent the time the file was created as an offset from midnight of Jan. 1, 1970. The second 8 bytes represent the time the file was modified as an offset from midnight of Jan. 1, 1970. Next are five 4-byte values (device, inode, mode, user id and group id) of file-attribute metadata related to the host OS. The only value used under Windows is the mode, which most often will be the octal 100644 I mentioned earlier when showing output from the `ls-files` command (this converts to the 4-byte 814AH value, which you can see at position 26H in **Figure 5**).

Following the metadata is the 4-byte length of the file contents. In **Figure 5**, this value starts at 030, which shows 00 00 0A 15 (2,581 decimal)—the length of the `.gitattributes` file on my system:

```
05/08/2017 09:24 PM <DIR> .
05/08/2017 09:24 PM <DIR> ..
05/08/2017 09:24 PM      2,581 .gitattributes
05/08/2017 09:24 PM      4,565 .gitignore
05/08/2017 09:24 PM <DIR> MSDNConsoleApp
05/08/2017 09:24 PM      1,009 MSDNConsoleApp.sln
                3 File(s)      8,155 bytes
                3 Dir(s) 92,069,982,208 bytes free
```

Figure 7 The Git Index File-Index Entry Data Format

IndexFile - Index Entry		
4 bytes	32-bit created time in seconds	Number of seconds since Jan. 1, 1970, 00:00:00.
4 bytes	32-bit created time - nanosecond component	Nanosecond component of the created time in seconds value.
4 bytes	32-bit modified time in seconds	Number of seconds since Jan. 1, 1970, 00:00:00.
4 bytes	32-bit modified time - nanosecond component	Nanosecond component of the created time in seconds value.
4 bytes	device	Metadata associated with the file—these originate from file attributes used on the Unix OS.
4 bytes	inode	
4 bytes	mode	
4 bytes	user id	
4 bytes	group id	
4 bytes	file content length	Number of bytes of content in the file.
20 bytes	SHA-1	Corresponding blob object's SHA-1 value.
2 bytes	Flags	(High to low bits) 1 bit: assume-valid/assume-unchanged flag 1-bit: extended flag (must be 0 for versions less than 3; if 1 then an additional 2 bytes follow before the path\file name) 2-bit: merge stage 12-bit: path\file name length (if less than 0xFFFF)
2 bytes (version 3 or higher)	Flags	(High to low bits) 1-bit: future use 1-bit: skip-worktree flag (sparse checkout) 1-bit: intent-to-add flag (git add -N) 13-bit: unused, must be zero
Variable Length	Path/file name	NUL terminated

At offset 034H is the 20-byte SHA-1 value for the blob object:
1ff0c423042b46cb1d617b81efb715defbe8054d.

Remember, this SHA-1 points to the blob object that contains the file contents for the file in question: `.gitattributes`.

At 048H is a 2-byte value containing two 1-bit flags, a 2-bit merge-stage value, and a 12-bit length of the path/file name for the current index entry. Of the two 1-bit flags, the high-order bit designates whether the index entry has its assume-unchanged flag set (typically done using the Git update-index plumbing command); the low-order bit indicates whether another two bytes of data precede the path\file name entry—this bit can be 1 only for index versions 3 and higher). The next 2 bits hold a merge-stage value from 0 to 3, as described earlier. The 12-bit value contains the length of the path\file name string.

If the extended flag was set, a 2-byte value holds the skip-worktree and intent-to-add bit flags, along with filler placeholders.

Finally, a variable length sequence of bytes contains the path\file name. This value is terminated with one or more NUL characters. Following that termination is the next blob object in the index or one or more index extension entries (as you'll see shortly).

Earlier, I mentioned that Git doesn't build tree objects until you commit what's been staged. What that means is the index starts

out with only path/file names and references to blob objects. As soon as you issue a commit, however, Git updates the index so it contains references to the tree objects it created during the last commit. If those directory references still exist in your working directory during the next commit, the cached tree object references can be used to reduce the work Git needs to do during the next commit. As you can see, the role of the index is multifaceted, and that's why it's described as an index, staging area and cache.

The index entry shown in **Figure 7** supports only blob object references. To store tree objects, Git uses an extension.

Index Extensions

The index can include extension entries that store specialized data streams to provide additional information for the Git engine to consider as it monitors files in the working directory and when it prepares the next commit. To cache tree objects created during the last commit, Git adds a tree extension object to the index for the working directory's root as well as for each sub-directory.

Figure 5, Marker 2, shows the final bytes of the index and captures the tree objects that are stored in the index. **Figure 8** shows the format for the tree-extension data.

The tree-extension data header, which appears at offset 284H, is composed of the string "TREE" (marking the start of the cached tree extension data) followed by a 32-bit value that indicates the length of the extension data that follows. Next are entries for each tree entry: The first entry is a variable-length null-terminated

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

CHICAGO
SEPT 18-21, 2017
DOWNTOWN MARRIOTT

**INTENSE TRAINING FOR DEVELOPERS, ENGINEERS,
PROGRAMMERS, ARCHITECTS AND MORE!**

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Native Client
- Software Practices
- Database and Analytics
- Angular JS
- ASP.NET / Web Server
- Agile
- ALM / DevOps
- Cloud Computing
- Windows Client
- Xamarin

**SWEET
127.0.0.1 CHICAGO**



**Register by August 18
and Save \$200!**

Use promo code VSLCHTI

**REGISTER
NOW**

EVENT PARTNER



GOLD SPONSOR



SUPPORTED BY



PRODUCED BY



YOUR GROWTH. OUR BUSINESS.

vslive.com/chicago

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

ANAHEIM, CA
OCT 16-19, 2017
HYATT REGENCY
A Disneyland® Good Neighbor Hotel



California CODIN'

**INTENSE TRAINING FOR DEVELOPERS, ENGINEERS,
PROGRAMMERS, ARCHITECTS AND MORE!**

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing



**Register by August 25
and Save \$300!**

Use promo code VSLANTI

**REGISTER
NOW**

EVENT PARTNER



SUPPORTED BY



PRODUCED BY



vslive.com/anaheim

string value for the tree path (or simply NUL for the root tree). The following value is an ASCII value, so it is to be read as the “7” you see in the hex editor—the number of blob entries covered by the current tree (because this is the root tree, it has the same number of entries you saw earlier when issuing the `Git ls-files` stage command). The next character is a space, followed by another ASCII number to represent the number of subtrees the current tree has.

The root tree for our project has only 1 subtree: `MSDNConsoleApp`. This value is followed by a line-feed character, then the SHA-1 for the tree. The SHA-1 starts at offset 291, beginning with `0d21e2`.

Let’s confirm that `0d21e2` is actually the root tree SHA-1. To do that, go to the command window and enter:

```
git log
This displays details of the recent commits:
commit 5192391e9f907eeb47aa38d1c6a3a4ea78e33564
Author: Jonathan Waldman <jonathan.waldman@live.com>
Date: Mon May 8 21:24:15 2017 -0500
```

Add project files.

```
commit dc0d3343fa24e912f08bc18aaa6f664a4a020079
Author: Jonathan Waldman <jonathan.waldman@live.com>
Date: Mon May 8 21:24:07 2017 -0500
```

Add `.gitignore` and `.gitattributes`.

The most recent commit is the one with the timestamp `21:24:15`, so that’s the one that last updated the index. I can use that commit’s SHA-1 to find the root-tree SHA-1 value:

```
git cat-file -p 51923
```

This generates the following output:

```
tree 0d21e2f7f60f77ead2cb85cc128efb13f56401d
parent dc0d3343fa24e912f08bc18aaa6f664a4a020079
author Jonathan Waldman <jonathan.waldman@live.com> 1494296655 -0500
committer Jonathan Waldman <jonathan.waldman@live.com> 1494296655 -0500
```

The preceding tree entry is the root tree object. It confirms that the `0d21e2` value at offset 291H in the index dump is, in fact, the SHA-1 for the root tree object.

The other tree entries appear immediately after the SHA-1 value, starting at offset 2A5H. To confirm the SHA-1 values for cached tree objects under the root tree, run this command:

```
git ls-tree -r -d master
```

This displays only the tree objects, recursively on the current branch:

```
040000 tree c7c367f2d5688dddc25e59525cc6b8ef0df914d MSDNConsoleApp
040000 tree 2723ceb04eda3051abf913782fadeebc97e0123c MSDNConsoleApp/Properties
```

The mode value of `040000` in the first column indicates that this object is a directory rather than a file.

Finally, the last 20 bytes of the index contain an SHA-1 hash representing the index itself: As expected, Git uses this SHA-1 value to validate the data integrity of the index.

While I’ve covered all of the entries in this article’s example index file, larger and more complex index files are the norm. The index file format supports additional extension data streams, such as:

- One that supports merging operations and merge-conflict resolution. It has the signature “REUC” (for resolve undo conflict).
- One for maintaining a cache of untracked files (these are files to be excluded from tracking, specified in the `.gitignore`

Figure 8 The Git Index File Tree-Extension Object Data Format

Index File - Cached Tree-Extension Header		
4 bytes	TREE	Fixed signature for a cached tree-extension entry.
4 bytes	32-bit number representing the length of TREE extension data	
Cached Tree-Extension Entry		
Variable	Path	NUL-terminated path string (null only for the root tree).
ASCII number	Number of entries	ASCII number representing the number of entries in the index covered by this tree entry.
1 byte	20H (space character)	
ASCII number	Number of subtrees	ASCII number representing the number of subtrees this tree has.
1 byte	0AH (linefeed character)	
20 bytes	Tree object’s SHA-1	SHA-1 values of the tree object this entry produces.

and `.git\info\exclude` files and by the file pointed to by `core.excludesfile`). It has the signature “UNTR.”

- One to support a split-index mode in order to speed index updates for very large index files. It has the signature “link.”

The index’s extension feature makes it possible to continue adding to its capabilities.

Wrapping Up

In this article, I reviewed the Git three-tree architecture and delved into details behind its index file. I showed you that Git updates the index in response to certain operations and that it also relies on information the index contains in order to carry out other operations.

It’s possible to use Git without thinking much about the index. Yet having knowledge about the index provides invaluable insight into Git’s core functionality while shedding light on how Git detects changes to files in the working directory, what the staging area is and why it’s useful, how Git manages merges, and why Git performs some operations so quickly. It also makes it easy to understand command-line variants of the check out and rebase commands—and the difference between soft, mixed and hard resets. Such features let you specify whether the index, working directory, or both the index and working directories should be updated when issuing certain commands. You’ll see such options when reading about Git workflows, strategies and advanced operations. The purpose of this article is to orient you to the important role the index plays so you can better digest the ways in which it can be leveraged. ■

JONATHAN WALDMAN is a Microsoft Certified Professional who has worked with Microsoft technologies since their inception and who specializes in software ergonomics. Waldman is a member of the Pluralsight technical team and he currently leads institutional and private-sector software-development projects. He can be reached at jonathan.waldman@live.com.

THANKS to the following Microsoft technical experts for reviewing this article: *Kraig Brockschmidt, Saeed Noursalehi, Ralph Squillace and Edward Thomson*

Actionable Messages for Outlook

Woon Kiat Wong

I love e-mail. At work, it's where I go to stay on top of what's going on and what I need to do. It's where I receive notifications of new expense reports submitted by my team, new replies to my tweets, new comments to my pull requests and so on. But e-mail could be so much better. Why do I need to click a link in e-mail and wait for the finance system Web site to load in a browser before I can approve an expense report? Why do I have to mentally change my context? I should be able to approve the expense report directly in the context of my e-mail client.

Sound familiar? Outlook is about to make your life much better, save you time and make you more productive.

Introducing Actionable Messages

Actionable Messages let users complete tasks within the e-mail itself. It offers a native experience in both the Outlook desktop client and Outlook Web Access (OWA). In this article, I'll use the word Outlook to mean either Outlook desktop client or OWA.

In the example I'll be using, the fictional company Contoso has an internal expense approval system. Every time an employee

submits an expense report, an e-mail message is sent to the manager for approval. I'll walk through the steps on how to use Actionable Messages in Outlook that lets the manager approve the request within the e-mail message itself.

My First Actionable Message

In **Figure 1**, you see the HTML of an Actionable Message. It might look complicated, but believe me, it's not. I'll explain the markup in detail in the following sections. The first step is to send an e-mail with the markup from **Figure 1** to your Office 365 e-mail account.

As shown in **Figure 2**, in the message itself, there's a message card with two buttons with which you can interact. If you click on the Approve button, it'll result in an error for now because you haven't yet specified the URL for the action. You'll add the URL later. If you click on the View Expense button, a browser will open and navigate to the Expense Approval Web site.

MessageCard Markup

The e-mail message itself is typical HTML markup. To make it an Actionable Message in Outlook, you insert MessageCard markup in the `<script>` element. One main advantage of this approach is that e-mail messages will continue to render as usual on clients that don't recognize the MessageCard markup. The format of this markup is called JSON-LD, which is a standard format to create machine-readable data across the Internet. Now, let's go through the markup in detail. These two lines of code are mandatory in every markup:

```
"@context": "http://schema.org/extensions",  
"@type": "MessageCard",
```

This article discusses:

- Create and send Actionable Messages
- Validating a JSON Web token
- Designing Web services for Actionable Messages

Technologies discussed:

Actionable Message Markup, Actionable Message JSON Web Token, Actionable Message Security



The Build vs. Buy **Data Quality** Challenge for Optimizing Accuracy

Q&A with Bud Walker, Vice President of Enterprise Sales & Strategy

Q Who is Melissa, and what solutions do you provide?

A Melissa Global Intelligence was founded in 1985 as Melissa Data. For over three decades, we have been a leading provider of data quality, ID verification, and data management solutions. Our software, cloud services, and data integration components leverage comprehensive and authoritative reference data to profile, verify, standardize, consolidate, match/dedupe, enrich, and update U.S. and global contact data, including names, addresses, phone numbers and email addresses for improved analytics, efficient operations, and strong customer relationships.

Q What are the most common causes of poor data quality?

A The major cause of bad data is from typographical errors and non-conforming data entered during the data entry process—either by employees or customers filling out contact forms. The next biggest cause of bad data comes from migrating data due to irregular, missing, or misplaced data values that cause surprises. Bad data costs businesses between 10-25% of revenue each year, and in 2016 cost US businesses over \$3.1 trillion. That's why implementing real-time validation of contact information like addresses, email, phone, and other important information is essential, as well as establishing a data governance team in charge of understanding the impact of data quality.

Q What is the difference between rules-based and active data quality, and why is active data quality so important?

A Data that is mostly static, internally generated and controlled—like KPIs—employee performance metrics, new product development, supplier payment optimization, and inventory reduction data, can usually be well managed by rules-based validation.

Active data like customer names, addresses, emails, phone numbers, company names, and job titles, constantly change and

require complex parsing rules and multisourced reference data for verification. For instance, the telephone number 949-555-5659 might look valid from a rules-based perspective, but is it actually callable, and associated with the right customer?

Active data quality relies on deep domain knowledge of contact and location data to parse, format, cleanse, enrich, and match customer records to provide the organization with accurate, timely, and actionable information.

Q Should companies build out a DQ solution or purchase an off-the-shelf one from a trusted vendor?

A We recommend a Hybrid approach. Companies should look to Build their own data quality solutions to handle rules-based data quality processes—those where they have the expertise and experience with their own internal data. Where active data quality is required, a Buy approach will usually be more efficient and cost-effective in the long run. Combining both Build and Buy can result in the best-of-both world results. We urge organizations to strive for small wins first—solve one problem at a time in discreet phases. This will help you show tangible results quickly and get the buy-in you need for future projects. Now, rinse and repeat.

Q How are Melissa's solutions employed?

A Melissa offers every kind of integration option you can imagine. We have on-prem APIs and Cloud services that allow you to build into existing or custom applications. We also offer plugins for data integration platforms like SQL Server®, Pentaho® and Talend®, and CRM software like Salesforce® and Dynamics® CRM. Our smart, sharp tools approach means we can help you create the best solutions based on your budget and needs and achieve data quality without breaking the bank.

For more information, please visit →

www.melissa.com

Figure 1 HTML of an Outlook Actionable Message

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf8">
<script type="application/ld+json">{
  "@context": "http://schema.org/extensions",
  "@type": "MessageCard",
  "hideOriginalBody": "true",
  "title": "Expense report is pending your approval",
  "sections": [
    {
      "text": "Please review the expense report below.",
      "facts": [
        {
          "name": "ID",
          "value": "98432019"
        }, {
          "name": "Amount",
          "value": "83.27 USD"
        }, {
          "name": "Submitter",
          "value": "Kathrine Joseph"
        }, {
          "name": "Description",
          "value": "Dinner with client"
        }
      ]
    },
    {
      "potentialAction": [
        {
          "@type": "HttpPost",
          "name": "Approve",
          "target": ""
        }, {
          "@type": "OpenUri",
          "name": "View Expense",
          "targets": [
            { "os": "default",
              "uri": "https://expense.contoso.com/view?id=98432019" }
          ]
        }
      ]
    }
  ]
}
</script>
</head>
<body>
<p>Please <a href="https://expense.contoso.com/view?id=98432019">approve</a>
expense report #98432019 for $83.27.</p>
</body>
</html>
```

You set the context to `http://schema.org/extensions` and the type to `MessageCard`. The `MessageCard` type indicates that this e-mail is an Actionable Message.

Next is the property `hideOriginalBody`. When the value is set to true, the e-mail body is hidden and only the card is displayed, as shown in Figure 2. This is useful when the card itself contains

all the information a user would need or the content of the card is redundant with the content of the e-mail body. In case the message is viewed in an e-mail client that doesn't understand message cards, then the original body will be shown and the message card will not, regardless of the value of `hideOriginalBody`. The value of the property `title` is the title of the `MessageCard`:

```
"hideOriginalBody": "true",
"title": "Expense report is pending your approval",
```

Next is `sections`. You can think of a section as representing an `activity`. If your card has multiple activities you should definitely use multiple sections, one per activity. Figure 3 shows markup with one section. You use the `facts` property of a section, which is an array of name-value pairs, to display the details of an expense report.

It would be great if you could visualize how the card looks when you're authoring the markup.

Next is `potentialAction`. This is an array of actions that can be invoked on this card. Currently the supported actions are `OpenUri` and `HttpPost`:

```
"potentialAction": [
  {
    "@type": "HttpPost",
    "name": "Approve",
    "target": ""
  }, {
    "@type": "OpenUri",
    "name": "View Expense",
    "targets": [
      { "os": "default",
        "uri": "https://expense.contoso.com/view?id=98432019" }
    ]
  }
]
```

The `OpenUri` action will open a browser and navigate to the URL specified in the `targets` property. The `targets` property is an array that lets you specify platform-specific URLs. For example, you might want users on iOS and Android to navigate to different URLs. In this example, you set the OS to default, which means the

URL is the same for all platforms.

The `HttpPost` action will make an HTTP POST request to an external Web service specified in the `target` property. Currently the value is empty. That's why you see an error when you click on the Approve button.

MessageCard Playground App

It would be great if you could visualize how the card looks when you're authoring the markup. Microsoft has a Web app that lets you do just that. It's called the `MessageCard Playground App` (bit.ly/2s274S9).

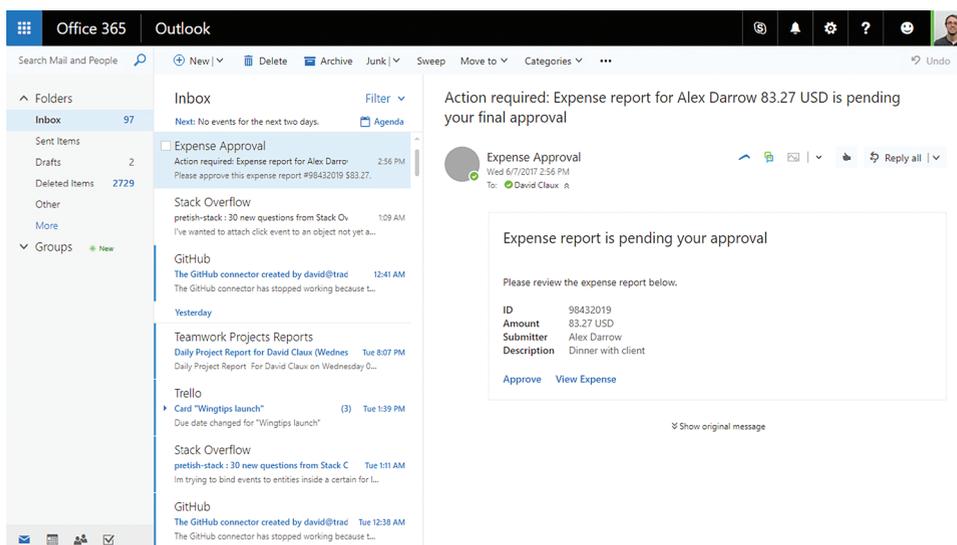
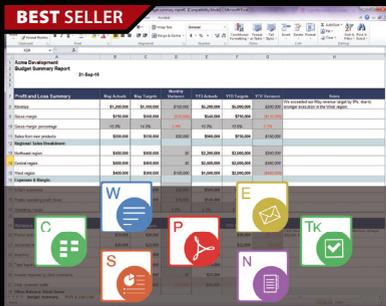


Figure 2 Actionable Message in Outlook Web Access

Experience the brand new look at www.componentsource.com



Aspose.Total for .NET from **\$2,939.02**



Create, Edit & Manipulate Word, Excel, PDF and PowerPoint Files in your .NET apps.

- Aspose.Total for .NET contains every Aspose .NET API in one complete package
- Also works with MS Visio, MS OneNote, MS Project, Email, Images, Barcodes, OCR & OMR
- Mail Merge, Document Assembly, Text Extraction, PST & OST Creation and File Conversion
- Create and Recognize Barcodes, Control Text Formatting, Paragraphs, Images and Tables
- Now includes the new Aspose.CAD and Aspose.3D APIs

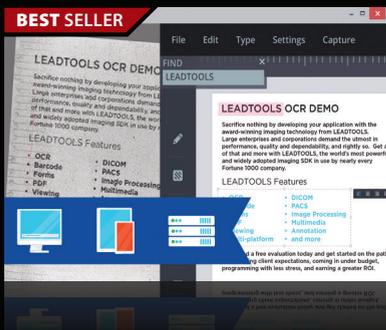


DevExpress DXperience 17.1 from **\$1,439.99**



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

- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context Toolbar controls
- CodeRush - New debug visualizer expression map and code analysis diagnostics



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



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

You should always design your card in the app first. Once you're happy with the card layout, you can then use the markup in your e-mail messages.

Calling an External Web Service with HttpPOST Action

Now you have a message card with two actions. The OpenUri will open a browser and navigate to the URL specified in the action. For the HttpPOST action, you'd like it to call your REST API that will approve the expense report. You replace the HttpPOST action with the following:

```
{
  "@type": "HttpPost",
  "name": "Approve",
  "target": "https://api.contoso.com/expense/approve",
  "body": "{ \"id\": \"98432019\" }"
}
```

When a user clicks on the Approve button, a Microsoft server will make an HTTP POST request that's similar to the following:

```
POST api.contoso.com/expense/approve
Content-Type: application/json

{ "id": "98432019" }
```

The target is the URL, which the Microsoft server is going to make a POST request to, and the body is the content of the request. The body content is always assumed to be JSON.

Now you'll send yourself an e-mail with the new markup. When you click on the Approve button, the action is completed successfully.

Figure 3 Card with One Section

```
"sections": [
  {
    "text": "Please review the expense report below.",
    "facts": [
      {
        "name": "ID",
        "value": "98432019"
      },
      {
        "name": "Amount",
        "value": "83.27 USD"
      },
      {
        "name": "Submitter",
        "value": "Jonathan Kiev"
      },
      {
        "name": "Description",
        "value": "Dinner with client"
      }
    ]
  }
],
```

Figure 4 ActionCard Action

```
"potentialAction": [
  {
    "@type": "HttpPost",
    ...
  },
  {
    "@type": "ActionCard",
    "name": "Reject",
    "inputs": [
      {
        "@type": "TextInput",
        "id": "comment",
        "isMultiline": true,
        "title": "Explain why the expense report is rejected"
      }
    ],
    "actions": [
      {
        "@type": "HttpPost",
        "name": "Reject",
        "target": "https://api.contoso.com/expense/reject",
        "body": "{ \"id\": \"98432019\", \"comment\": \"{{rejectComment.value}}\" }"
      }
    ]
  },
  {
    "@type": "OpenUri",
    ...
  }
]
```

ActionCard Action

Now let's add a Reject button so users can reject an expense report. For reject, you need additional input from users to explain why the expense report is rejected.

The ActionCard action is designed for such scenarios. It contains one or more inputs and associated actions that can be either OpenUri or HttpPost. You insert an ActionCard action in between HttpPOST and OpenUri, as shown in **Figure 4**.

Actionable Messages will work with any Web service that can handle HTTP POST requests.

If you send yourself the updated markup, there are Approve, Reject and View Expense buttons. If you click on the Reject button, you can now enter comments before you reject the expense report.

Let's take a look at the ActionCard action markup. Besides the type and name properties, it has an array of inputs and actions. In this example, you have a multiline TextInput that lets users enter text. The other supported inputs are DateTime and MultiChoice-Input. For more details, refer to bit.ly/2t3bLJN.

You have an HttpPOST action that will make a call to the external Web service to reject the expense report. This is similar to the HttpPOST action for the approve action. One major difference is that you want to pass the comments entered by users to the Web service call. You can reference to the value of the text input by using `{{rejectComment.value}}`, where `rejectComment` is the ID of the text input.

Web Service for Actionable Messages

So far you've seen the markup for Actionable Messages in Outlook and how it works. In the rest of the article, I'll describe how a Web service should handle requests coming from Actionable Messages in Outlook.

Actionable Messages will work with any Web service that can handle HTTP POST requests. In this example, your Web service is an API controller in ASP.NET MVC. **Figure 5** shows your API controller.

There are two methods in this API controller, one for approval and another for rejection. The Web service must return an HTTP status code of 2xx for the action to be considered successful. The Web service can also include the `CARD-ACTION-STATUS` header in the response. The value of this header will be displayed to the user in a reserved area of the card. If you deploy the Web service to `https://api.contoso.com` and you click on the Approve button, you'll get the notification that the operation was completed successfully, as shown in **Figure 6**.

You now have the Actionable Message working end to end. You can send out the Actionable Message and when the user clicks on the Approve button, an HTTP POST request is made to your Web service. Your Web service will process the request and return 200 OK. Outlook will then mark the action as done. Next, I'll look at how you can secure your Web service.

Figure 5 Expense API Controller

```
[RoutePrefix("expense")]
public class ExpenseController : ApiController
{
    [HttpPost]
    [Route("approve")]
    public HttpResponseMessage Approve([FromBody]JObject jBody)
    {
        string expenseId = jBody["id"].ToString();

        // Process and approve the expense report.
        HttpResponseMessage response = this.Request.CreateResponse(HttpStatusCode.OK);
        response.Headers.Add("CARD-ACTION-STATUS", "The expense was approved.");

        return response;
    }

    [HttpPost]
    [Route("reject")]
    public HttpResponseMessage Reject([FromBody]JObject jBody)
    {
        string expenseId = jBody["id"].ToString();
        string comment = jBody["comment"].ToString();

        // Process and reject the expense report.
        HttpResponseMessage response = this.Request.CreateResponse(HttpStatusCode.OK);
        response.Headers.Add("CARD-ACTION-STATUS", "The expense was rejected.");

        return response;
    }
}
```

Limited-Purpose Tokens

Because the expense ID usually follows a certain format, there's a risk that an attacker can perform an attack by posting a lot of requests with different expense IDs. If an attacker successfully guesses an expense ID, the attacker might be able to approve or reject that expense report. Microsoft recommends developers use "limited-purpose tokens" as part of the action target URL or in the body of the request. The limited-purpose token should be hard for attackers to guess. For example, I use a GUID, a 128-bit number as the limited-purpose token. This token can be used to correlate service URLs with specific requests and users. It can also be used to protect Web services from replay attacks (bit.ly/2sBQmdn). You update the markup to include a GUID in the body:

```
{
  "@type": "HttpPost",
  "name": "Approve",
  "target": "https://api.contoso.com/expense/approve",
  "body": "{ \"id\": \"98432019\", \"token\": \"d8a0bf4f-ae70-4df6-b129-5999b41f4b7f\" }"
}
```

Bearer Token

While limited-purpose tokens make it harder for attackers to forge a request, they're still not perfect. Ideally, a Web service should be able to tell whether an HTTP POST request is coming from a Microsoft server instead of some unauthorized, potentially malicious server.

Microsoft solves this problem by including a bearer token in every HTTP POST request it sends to Web services. The bearer token is a JSON Web Token (JWT) and it's included in the Authorization header of a request. When a user clicks on the Approve button, the Web service will receive a request that looks like this:

```
POST https://api.contoso.com/expenses/approve

Content-Type: application/json
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIngldCI6IjhxZ3A4VERCbdJlNkp5RkU0WjMOZDJoYS1rRSIsImtpZCI6IjhxZ3A4VERCbdJlNkp5RkU0WjMOZDJoYS1rRSJ9.eyJpYXQiOiE0ODQwODkyNzksInZlciI6IjNUSS5FeHRIcm5hbEFjY2VzclRva2VuL1YxIiwiaXNwYWQiOiI0OGFmMDhkYy1mNmQyLTQzNWYtYjJhNy0wNj1hYmQ5OWMwODYiLCJzdWIiOiJkYXZpZEBjb250b3NvLmNvbSIsImFwcG1kYm91IjoiaSImFjciI6IjAxLCJzZW5kZXI6IjEhbnNlYXBw... (truncated for brevity)
```

```
{
  "id": "98432019",
  "token": "d8a0bf4f-ae70-4df6-b129-5999b41f4b7f"
}
```

What follows "Bearer" in the Authorization header is a long base-64 encoded string that's a JSON Web Token (JWT). You can decode the JWT at jwt.calebb.net. **Figure 7** shows a sample token after it's decoded.

Every JWT has three segments, separated by a dot (.). The first segment is the header, which describes the cryptographic operations applied to the JWT. In this case, the algorithm (alg) used to sign the token is RS256, which means RSA using the SHA-256 hash algorithm. The x5t value specifies the thumbprint of the key used to sign the token.

The second segment is the payload itself. It has a list of claims that the token is asserting. Web services should use these claims to verify a request. The table in **Figure 8** describes these claims.

The third segment is the digital signature of the token. By verifying the signature, Web services can be confident that the token is sent by Microsoft and trust the claims in the token.

Verifying a digital signature is a complex task. Fortunately, there's a library on NuGet that makes the verification task easy. The library is available at bit.ly/2stq90c and it's authored by Microsoft. Microsoft also published code samples for other languages on how to verify the token. Links for these code samples are available at the end of this article.

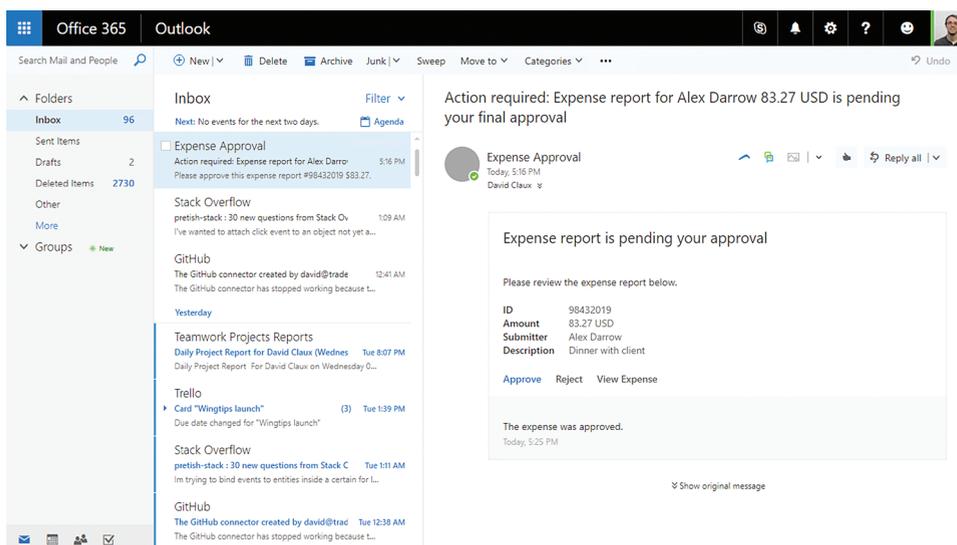


Figure 6 Expense Report with Successful Approval Notification

WE'RE CHANGING THE WAY YOU LOOK AT REPORTING

The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor.

Our award-winning developer libraries are completely independent from MS Word or any other third-party application and can be completely integrated into your business applications.

Follow the trend and switch to flow type layout reporting.

www.textcontrol.com

www.reporting.cloud



WINDOWS FORMS



WPF



ASP.NET MVC



ASP.NET AJAX



CLOUD WEB API

TEXT CONTROL

REPORTING LIBRARIES FOR WINDOWS, WEB, MOBILE AND CLOUD APPLICATIONS



Figure 7 A Sample Bearer JSON Web Token

```
{
  typ: "JWT",
  alg: "RS256",
  x5t: "8qgp8TDB12H6JyFE4Z34d2ha-kE",
  kid: "8qgp8TDB12H6JyFE4Z34d2ha-kE"
}.
{
  iat: 1484089279,
  ver: "STI.ExternalAccessToken.V1",
  appid: "48af08dc-f6d2-435f-b2a7-069abd99c086",
  sub: "david@contoso.com",
  appidacr: "2",
  acr: "0",
  sender: "expenseapproval@contoso.com",
  iss: "https://substrate.office.com/sts/",
  aud: "https://api.contoso.com",
  exp: 1484090179,
  nbf: 1484089279
}.
[signature]
```

After you include the NuGet package in the Web service project, you can use the `VerifyBearerToken` method, as shown in Figure 9, to verify the bearer token in a request.

Verifying a digital signature is a complex task. Fortunately, there's a library on NuGet that makes the verification task easy.

First, the method verifies there's a bearer token in the `Authorization` header. Then, it initializes a new instance of `ActionableMessageTokenValidator` and calls the `ValidateTokenAsync` method. The method takes two parameters. The first one is the bearer token itself. The second one is the Web service base URL. If you look at the decoded JWT, this is the value

Figure 8 Description of Claims in Payload

Claims	Description
iss	The token issuer. The value should always be <code>https://substrate.office.com/sts/</code> . The Web service should reject the token and the request if the value does not match.
appid	The ID of the application which issues the token. The value should always be 48af08dc-f6d2-435f-b2a7-069abd99c086 . The Web service should reject the token and the request if the value doesn't match.
aud	The audience of the token. It should match the hostname of the Web service URL. The Web service should reject the token and the request if the value doesn't match.
sub	The subject who performed the action. The value will be the e-mail address of the person who performed the action, if the e-mail address or any of the proxy e-mail addresses is in the <code>To</code> : line. If none of the e-mail addresses is matched, this will be the hashed value of the subject's user principal name (UPN). It's guaranteed to be the same hashed value for the same UPN.
sender	The e-mail address of the original message sender.
tid	The tenant ID of the token issuer.

of the `aud` (audience) claim. It basically means the token is issued for the intended audience, which is your Web service but not any other Web service. In this case, the API to be called is `http://api.contoso.com/expense/approve`. The value in the claim will be the base URL, which is `https://api.contoso.com`.

The method will return an instance of `ActionableMessageTokenValidationResult`. First, you'll check the property

Figure 9 The `VerifyBearerToken` Method

```
private async Task<HttpStatusCode> VerifyBearerToken(
    HttpRequestMessage request, string serviceBaseUrl, string expectedSender)
{
    if (request.Headers.Authorization == null ||
        !string.Equals(request.Headers.Authorization.Scheme, "bearer",
            StringComparison.OrdinalIgnoreCase) ||
        string.IsNullOrEmpty(request.Headers.Authorization.Parameter))
    {
        return HttpStatusCode.Unauthorized;
    }

    string bearerToken = request.Headers.Authorization.Parameter;
    ActionableMessageTokenValidator validator =
        new ActionableMessageTokenValidator();
    ActionableMessageTokenValidationResult result =
        await validator.ValidateTokenAsync(bearerToken, serviceBaseUrl);

    if (!result.ValidationSucceeded)
    {
        return HttpStatusCode.Unauthorized;
    }

    if (!string.Equals(result.Sender, expectedSender,
        StringComparison.OrdinalIgnoreCase) ||
        !result.ActionPerformer.EndsWith("@contoso.com",
            StringComparison.OrdinalIgnoreCase))
    {
        return HttpStatusCode.Forbidden;
    }

    return HttpStatusCode.OK;
}

[HttpPost]
[Route("approve")]
public async Task<HttpResponseMessage> Approve([FromBody]JObject jBody)
{
    HttpRequestMessage request = this.ActionContext.Request;
    HttpStatusCode result = await VerifyBearerToken(
        request, "https://api.contoso.com",
        "expenseapproval@contoso.com");

    switch (result)
    {
        case HttpStatusCode.Unauthorized:
            return request.CreateErrorResponse(
                HttpStatusCode.Unauthorized, new HttpError());

        case HttpStatusCode.Forbidden:
            HttpResponseMessage errorResponse =
                this.Request.CreateErrorResponse(HttpStatusCode.Forbidden, new HttpError());
            errorResponse.Headers.Add("CARD-ACTION-STATUS",
                "Invalid sender or the action performer is not allowed.");
            return errorResponse;

        default:
            break;
    }

    string expenseId = jBody["id"].ToString();

    // Process and approve the expense report.

    HttpResponseMessage response = this.Request.CreateResponse(HttpStatusCode.OK);
    response.Headers.Add("CARD-ACTION-STATUS", "The expense was approved.");

    return response;
}
```

ValidationSucceeded. If the validation succeeded, the value will be true; otherwise, it'll be false.

The result also includes two other properties that will be useful to third parties. The first one is Sender. This is the value of the sender claim in the token. This is the e-mail address of the account that sent the actionable message. The second one is the Action-Performer, which is the value of the sub claim. This is the e-mail address of the person who performed the action. In this example, only those with @contoso.com e-mail addresses can approve or reject an expense report. You can replace the code with a more complicated verification of your own.

Refresh Card

So far the only way to provide feedback to a user is through the CARD-ACTION-STATUS header. The value of the header will be displayed to the user in a reserved area of the card. Another option is to return a refresh card to the user. The idea is to replace the current action card with a different card. There are a few reasons why you want to do that. For example, after an expense report is approved, you don't want users to be able to approve or reject the expense report again. Instead, you'd like to tell the user that the expense report is already approved. **Figure 10** shows the markup that you'll return.

You need to set the value of the header CARD-UPDATE-IN-BODY to true so Microsoft servers know that the response has a refresh card. **Figure 11** shows the Approve method returns a refresh card.

Wrapping Up

Actionable Messages let users complete tasks within Outlook in a secure way. It's available in desktop Outlook and Outlook Web Access today and the feature is coming to Outlook for Mac and Outlook Mobile soon. It's straightforward to implement Actionable Messages. First, you need to add the required markup to the e-mails you're sending out. Second, you need to verify the bearer token sent by Microsoft in your Web service. Actionable Messages will make your users happier and more productive. There is so much more about Actionable Messages than this article can cover. Visit bit.ly/2rAD6AZ for the complete references and links to the code samples.

Figure 10 Markup Returned to Expense Report with Refresh Card

```
{
  "@context": "http://schema.org/extensions",
  "@type": "MessageCard",
  "hideOriginalBody": "true",
  "title": "Expense report #98432019 was approved",
  "sections": [
    {
      "facts": [
        {
          "name": "ID",
          "value": "98432019"
        },
        {
          "name": "Amount",
          "value": "83.27 USD"
        },
        {
          "name": "Submitter",
          "value": "Kathrine Joseph"
        },
        {
          "name": "Description",
          "value": "Dinner with client"
        }
      ]
    }
  ]
}
```

Figure 11 The Approve Method Returns a Refresh Card

```
private HttpResponseMessage CreateRefreshCard(
    HttpRequestMessage request, string actionStatus,
    string expenseID, string amount, string submitter, string description)
{
    string refreshCardFormatString = @"{@context\": \"http://schema.
    org/extensions\", \"@type\": \"MessageCard\", \"hideOriginalBody\":
    \"true\", \"title\": \"Expense report #{0} was approved\", \"sections\":
    [ { \"facts\": [ { \"name\": \"ID\", \"value\": \"{0}\", { \"name\":
    \"Amount\", \"value\": \"{1}\", { \"name\": \"Submitter\", \"value\":
    \"{2}\", { \"name\": \"Description\", \"value\": \"{3}\" } ] ] } ] }";
    string refreshCardMarkup = string.Format(
        refreshCardFormatString,
        expenseID,
        amount,
        submitter,
        description);

    HttpResponseMessage response = request.CreateResponse(HttpStatusCode.OK);
    Response.Headers.Add("CARD-ACTION-STATUS", actionStatus);
    response.Headers.Add("CARD-UPDATE-IN-BODY", "true");
    response.Content = new StringContent(refreshCardMarkup);

    return response;
}

[HttpPost]
[Route("approve")]
public async Task<HttpResponseMessage> Approve([FromBody]JObject jBody)
{
    HttpRequestMessage request = this.ActionContext.Request;
    HttpStatusCode result = await VerifyBearerToken(
        request, "https://api.contoso.com",
        "expenseapproval@contoso.com");

    switch (result)
    {
        case HttpStatusCode.Unauthorized:
            return request.CreateErrorResponse(
                HttpStatusCode.Unauthorized, new HttpError());

        case HttpStatusCode.Forbidden:
            HttpResponseMessage errorResponse =
                this.Request.CreateErrorResponse(
                    HttpStatusCode.Forbidden, new HttpError());
            errorResponse.Headers.Add("CARD-ACTION-STATUS",
                "Invalid sender or the action performer is not allowed.");
            return errorResponse;

        default:
            break;
    }

    string expenseId = jBody["id"].ToString();

    // Process and approve the expense report.

    return CreateRefreshCard(
        request,
        "The expense was approved.",
        "98432019",
        "83.27 USD",
        "Jonathan Kiev",
        "Dinner with client");
}
```

I'd like to acknowledge Sohail Zafar, Edaena Salinas Jasso, Vasant Kumar Tiwari, Mark Encarnacion and Miaosen Wang, who helped review this article for grammar, spelling, and flow. ■

Woon Kiat Wong is a software engineer from the Knowledge Technologies Group in Microsoft Research. He works closely with the Outlook team to deliver Actionable Messages. Contact him at wowong@microsoft.com.

THANKS to the following Microsoft technical experts for reviewing this article: Pretish Abraham, David Claux, Mark Encarnacion and Patrick Pantel

Batch Processing Using a Serverless Architecture

Joseph Fultz

For Azure enterprise customers, a key challenge in managing their cloud footprint is the ability to control what they spend and to charge those costs back to the consumers. Fortunately, there are several vendors that provide tools, such as Cloud Cruiser, Cloudyn, and Cloudability, to help with collecting usage data and generating a rich set of reports. Additionally, you can find many good examples of how to pull data programmatically, such as the post from a former co-worker of mine, Ed Mondek, in which he shows how to pull data into Excel and view it (bit.ly/2rzDOPI). However, if you want to pull that data regularly and enable historical, present trend and predictive views, you need to store a lot more data. For a large enterprise with thousands of resources per subscription, that amount of data can be daunting and is certainly not what you'd want to fetch and keep on a local machine.

Luckily, there's another way. In this article I'm going to walk you through the serverless Extract-Transform-Load (ETL) process I set up to extract such data, provide a little enrichment and store the data to a place where further work (analytics, map-reduce and so forth)

can be done. I'll touch on the overall design, key decision points and important things to consider in taking a serverless approach.

Determining Consumption

The first decision is choosing between the Enterprise Agreement (EA) Billing API and the Azure Billing API, which centers its requests around specific subscriptions. My prototype is targeted at enterprise customers with multiple enrollments in an EA. In the scenario with which I'm working, subscriptions are being used as part of the management boundaries for both specific product groups and for separating production from non-production resources. This could result in a fairly high number of subscriptions in flux due to the volatile proof-of-concept (PoC) type of work being created as new groups and new product lines start up in Azure. Thus, I chose to work with the EA API because it reduced the scope of work in that I don't have to create a discovery mechanism for subscriptions. This leaves me with the noted challenge of not having any data for subscriptions created outside of the enrollments for the enterprise. While this is an important area to tackle, it comes with a number of other process and management challenges that have to be solved organizationally and is outside the scope of the work I want to accomplish.

Requirements and Logical Flow

In any architecture, it's the intersections between systems that require the most scrutiny in design and testing. A serverless architecture doesn't change the need to consider the volume of data that moves

This article discusses:

- Design considerations in a serverless architecture
- Integrating multiple Azure Platform-as-a-Service capabilities
- Billing data retrieval

Technologies discussed:

Azure CosmosDB, Azure Functions, Azure Blob Storage

through the supersystem, and must take into account the particular constraints of the discrete subsystems. The principal change in architecting such a supersystem is more in the depth or scope when defining the system, such as sizing a queue for throughput, but not sizing the hardware that hosts it. You must still consider latency, connectivity, volume, availability, cost, and any number of other factors, but the work of sizing and defining the particulars of the service ends once you've defined the capacity and the cost of the capability needed to meet the identified requirements. There's no additional work of defining the host environment and all its needed artifacts as you might have done in the past.

Before I get into designing what the overall flow of information into the system will look like, let's note a few facts about the source systems and some requirements for the end-state system:

- All of the data for every subscription under the EA will be returned for all resources for every day it's available in the designated month. This can result in a lot of data, with a linear growth as the month progresses.
- Any and all records may be updated throughout the month. The stated settlement timing is 72 hours. As a point of safety, I'll consider all records in flux for a given month until 72 hours past the beginning of the subsequent month.
- The usage data isn't returned with an ID for the enrollment, so I'll have to add it.
- Determining cost is a separate activity and requires retrieving the rate card and further processing.
- No information will be received for subscriptions that aren't in the specified EA.

Additionally, there are a few technical business requirements that the prototype must include:

- The ability to create read-only and geographically distributed datasets must be included.
- Processing performance should be adjustable for cost versus performance.
- The ability to secure access at the subscription level should be designed in.

The overall flow itself is fairly simple in that I'm simply going to retrieve the data, add a small amount of information and persist it into the target storage.

As depicted in **Figure 1**, the path for getting the data to its target is fairly simple because there's no integration with any external systems other than the EA Billing API. I know that when I work through the data, I'll have to do some amount of initial processing and enrichment (for example, add the enrollment ID), and on the persistence side I'll have to deal with existing records from the previous day's fetches. I'll probably want to look at separating those two processes.

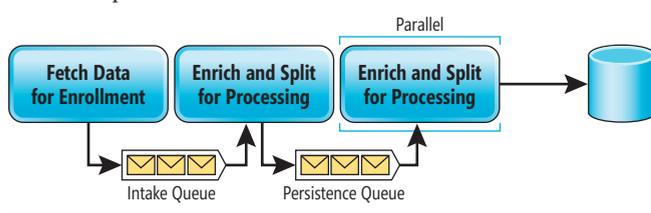


Figure 1 Logical Flow

Thus, you see three major blocks that represent retrieval, enrichment and persistence, which are all separated by some queuing mechanism. The complications start after I make some technology picks and start looking at the details of implementing with those components and making the processing pipeline run in parallel.

Technology Mapping

At this point in the process, two factors beyond the requirements of the overall system may come into play: enterprise standards and personal preference. If these are in conflict, the result can be almost endless debate. Fortunately, in this instance I don't have to worry about this. I do have my own mix of constraints, along with those I noted from the initial requirements. In this case, I'd like to make sure to hit these marks:

- Simplest compute provisioning and edits/updates for quick cycles of testing
- Easy automatic scaling
- Easy provisioning for geographic distribution of data
- Easy mechanisms for scheduling and triggering work

Here, I want to focus on the work and not on the system setup. I'll leave things like cost analysis for various implementations and adherence to corporate standards until after I have a working prototype. I did consider some alternatives, such as Azure SQL Database versus Azure Cosmos DB, but I'm going to focus on my choices and the primary motivations for each of those choices.

- Compute: Azure Functions will serve me well here. It meets my need for scale and simplicity while also providing easy configuration of scheduled and triggered jobs and easy integrations with bindings.
- Queuing: Keeping things simple, I'll use Azure Storage Blobs and separate the files by containers. The unknown but expectedly large size of each initial input file makes storage queues a non-option for initial retrieval, and likely takes them out of the running for processing individual subscription data splits. Beyond that, I'd like to keep the mechanism uniform and I really don't need any advanced capabilities, such as priority messages, routing, message-specific security and poisoned message handling.
- Storage: Azure Cosmos DB is indeed my friend here. Using the subscription ID as the partition key allows me to limit access by subscription, if necessary. Additionally, the ease of adding and removing read and read-write geographically distributed replicas and native support in Power BI makes this a no-brainer for my system. Last, I have to admit a little personal bias: I want a proper document storage mechanism that supports the SQL syntax I've used for too many years to abandon.

Figure 2 represents the application of technology to the logical architecture, as well as adding some processing flow to it.

I've taken the liberty of including the names I used in this diagram, but you might not have names at this stage of the design. The shapes used indicate the technology in play; the numbers on the line are the sequence in which the process is executed, and the arrows indicate which component initiates the outbound call. Note that I've identified four Azure Functions, four Azure Storage

Blob Containers and three Azure Cosmos DB collections that I'll employ as the working pieces of my implementation.

Separating the data into three collections is useful for explaining, but serves a grander purpose. I won't need the same security for each of the types of documents and the separation makes that easy to understand and manage. More important, I define the performance characteristics by collection and the separation allows me to more easily optimize that by having a large high-throughput collection specifically for the DetailedUsageData, while the other two remain minimal.

Retrieving Data

Starting with the first two legs of the data journey, I want to run something similar to what I do with a Cron job. While the WebJobs SDK itself would support this type of implementation, it would leave a lot of work of configuring the runtime environment to me and increase my overall development effort. Because Azure Functions is built on top of the WebJobs SDK and naturally supports Timer Trigger, it's an easy choice. I could've used Azure Data Factory because it's a tool made specifically for moving data around and it supports retrieving Web data and working with Blobs. However, that would mean I'd need to work out certain things with regard to reference data and updating duplicate records in Azure Cosmos DB when I don't have the row ID. Familiarity with development and debugging using Azure Functions, and the information I can get from Azure Functions integration with Application Insights, makes Azure Functions my preferred choice in this instance.

The Timer Trigger has an obvious function, but in order for DailyEABatchControl to know what to process, it retrieves configuration information from the Enrollments collection, which has the following schema:

```
{
  "enrollmentNumber": "<enrollment number>",
  "description": "",
  "accessKey": "<access key>",
  "detailedEnabled": "true",
  "summaryEnabled": "false",
}
```

For now, having the enrollment number, access key and a flag to turn on processing ("detailedEnabled") is sufficient for me to do work. However, should I start adding capabilities and need additional run configuration information, Azure Cosmos DB will allow me to easily add elements to the document schema without having to do a bunch of reworking and data migration. Once the DailyEABatchControl is triggered, it will loop through all of the documents and call RetrieveUsage for each enrollment that has "detailedEnabled" set to true, separating the logic to start a job from the logic to retrieve the source data. I use the JobLog

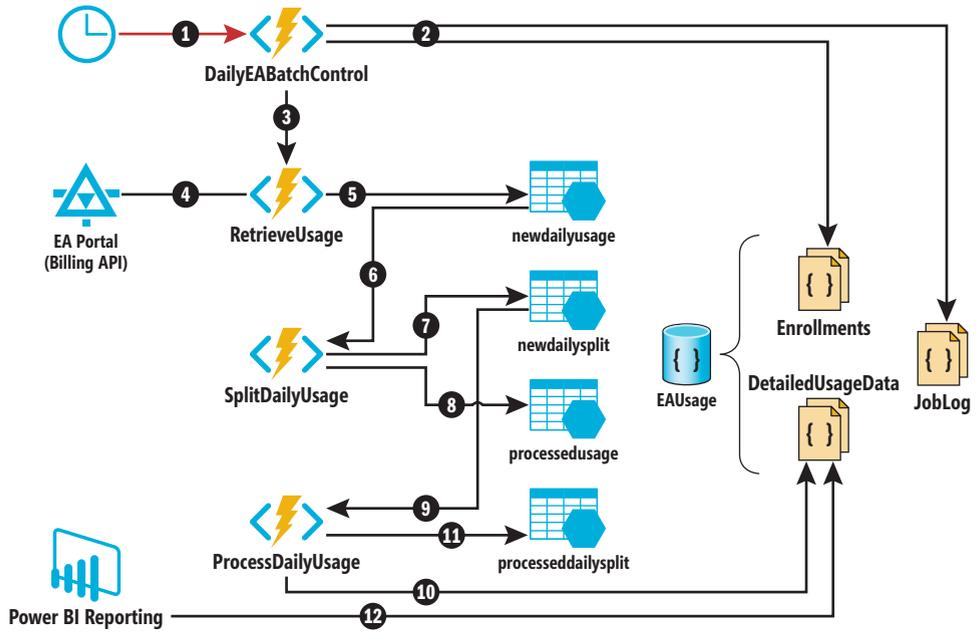


Figure 2 Technology Map and Data Flow

collection to determine if a job has already been run for the day, as shown in Figure 3.

The last lambda results in a filtered list of enrollments for which data hasn't been retrieved for the day in question. Next, I'll call the RetrieveUsage (step 3 in Figure 2) from within DailyEABatchControl by calling it with HttpClient with sufficient data in the post body for it to know the enrollment for which it's fetching data and the month for which it's fetching it, as shown in Figure 4.

It's worth pointing out that this isn't intended to be an open system. I'm creating a closed processing loop so I don't want just any caller executing the RetrieveUsage Function. Thus, I've secured it by requiring a code that's not shown in Figure 4, but is part of the URI returned from GetEnvironmentVariable("retrieveUsageUri"). In an enterprise implementation, a service principal and Azure Active Directory integration would be a more realistic choice to achieve a higher degree of security.

Figure 3 Job Control Logic

```
// Get list of enrollments for daily processing
List<Enrollment> enrollments =
    inputDocument.CreateDocumentQuery<Enrollment>(
        UriFactory.CreateDocumentCollectionUri(dbName, enrollmentCollection),
        new SqlQuerySpec("SELECT * FROM c WHERE c.detailedEnabled = 'true'"),
        queryOptions).ToList<Enrollment>();

// Get yesterday's date to make sure there are logs for today
int comparisonEpoch =
    (int)(DateTime.UtcNow.AddDays(-1) - new DateTime(1970, 1, 1)).TotalSeconds;

string logQuery =
    "SELECT * FROM c WHERE c.epoch > " + comparisonEpoch.ToString() + "";

List<JobLog> logs = inputDocument.CreateDocumentQuery<JobLog>(
    UriFactory.CreateDocumentCollectionUri(dbName, jobLogCollection),
    new SqlQuerySpec(logQuery), queryOptions).ToList<JobLog>();

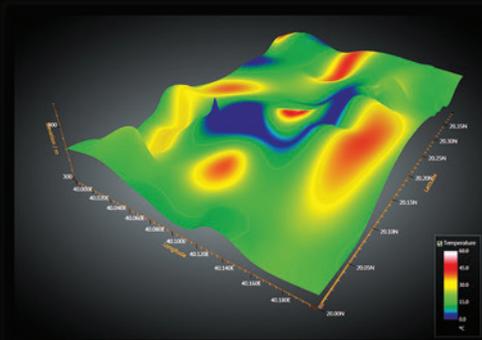
// Get list of enrollments for which there is no match
var jobList = enrollments.Where(x =>
    !logs.Any (l => l.enrollmentNumber == x.enrollmentNumber));
```



- [WPF]
- [Windows Forms]
- [Free Gauges]
- [Data Visualization]
- [Volume Rendering]
- [3D / 2D Charts] [Maps]

LightningChart[®]

The fastest and most advanced charting components



Create **eye-catching** and **powerful** charting applications for engineering, science and trading

- DirectX GPU-accelerated
- Optimized for real-time monitoring
- Supports gigantic datasets
- Full mouse-interaction
- Outstanding technical support
- Hundreds of code examples

NEW

- Now with Volume Rendering extension
- Flexible licensing options

Get free trial at
LightningChart.com/ms



Figure 4 Retrieving Usage Data

```
foreach(var doc in jobList)
{
    HttpClient httpClient = new HttpClient();

    string retrieveUsageUri = @"https://" +
        System.Environment.GetEnvironmentVariable("retrieveUsageUri");

    string postBody = "{ \"enrollment\": \"\" + doc.enrollmentNumber + "\", \" +
        \"month\": \"\" + DateTime.Now.ToString(\"yyyy-MM\") + \"\" }";

    httpClient.DefaultRequestHeaders.Accept.Add(
        new MediaTypeWithQualityHeaderValue("application/json"));

    var content = new StringContent(postBody, Encoding.UTF8, "application/json");
    var response = await httpClient.PostAsync(theUri, content);

    response.EnsureSuccessStatusCode();

    string fetchResult = await response.Content.ReadAsStringAsync();
}
```

The last step of the first leg of my data's journey is within the `RetrieveUsage` function, where it's persisted to the `newdailyusage` container with Azure Blob Storage. However, in order to get that data I have to construct the call and include the `accessKey` as a bearer token in the header:

```
HttpClient httpClient = new HttpClient();

string retrieveUsageUri = usageQB.FullEARreportUrl();

httpClient.DefaultRequestHeaders.Add("authorization", bearerTokenHeader);
httpClient.DefaultRequestHeaders.Add("api-version", "1.0");

var response = await httpClient.GetAsync(retrieveUsageUri);

response.EnsureSuccessStatusCode();

string responseText = await response.Content.ReadAsStringAsync();
```

For the sake of brevity, I've cut some date manipulations out of this code block and haven't included a helper class for generating the `bearerTokenHeader` or the `UsageReportQueryBuilder`. However, this should be sufficient to illustrate how they're used and ordered. The `accessKey` is passed into the static method `FromJwt`, which will return the `BearerToken` type, from which I simply grab the header and add it to the request that's created from the URL constructed by the call to `usageQB.FullEARreportUrl`. Last, I update the output binding to the path and filename I want for the Blob target:

```
path = "newdailyusage/" + workingDate.ToString("yyyyMMdd")
    + "-" + data.enrollment + "-usage.json";
var attributes = new Attribute[]
{
    new BlobAttribute(path),
    new StorageAccountAttribute("eabillingstorage_STORAGE")
};

using (var writer = await binder.BindAsync<TextWriter>(attributes))
{
    writer.Write(responseText);
}
```

This will result in a structure in Azure Storage that looks like this:

```
newdailyusage/
    20170508-1234-usage.json
    20170508-456-usage.json
    20170507-123-usage.json
```

This allows me to store data multiple enrollments and multiple files for each enrollment in case processing doesn't happen for some reason. Additionally, because data can change for previous days as the month progresses, it's important to have the files available for research and reconciliation in case anomalies show up in the report data.

Splitting Data for Parallel Processing

With so much data coming in and the work of somehow updating records for a given month of processing each day, it's important to process this data in a parallel fashion. Usually, at least nowadays, this is when I break out the parallel libraries for C#, write a few lines of code and pat myself on the back for being a genius at parallel processing. However, in this instance, I'd really like to just rely on the capabilities of the platform to do that for me and allow me to focus on each discrete task.

The next Azure Function in the sequence has been configured with a blob trigger so it will pick up files that land in the inbound processing storage container. The job at this step is to split the inbound file into a file-per-day per enrollment. All in all, this is a pretty simple step, but it does require deserializing the JSON file into RAM. It's important to note this, because the method I've chosen to use for the prototype simply calls the `deserialize` method:

```
JsonConvert.DeserializeObject<List<EAUsageDetail>>(myBlob);
```

I know this to be sufficient for my purposes, but the present RAM allocation for the Azure Function host is 1.5GB. It's possible that, for a large enrollment with substantial resources provisioned, a file would become too big at some point in the month to load into RAM, in which case an alternate method for parsing and splitting the file will have to be used. Moreover, if you create an Azure Function that takes more than five minutes to run, it will have to be modified because the current default is five minutes, though this can be adjusted to a max of 10 minutes via the host configuration JSON. As I mentioned early on, knowing the volume of data will be key at each point and for integration in the overall system. Once the data has been deserialized, I'll grab the max day out of it and set up a loop from day one to day max to start selecting out the data for each of those days, as shown in **Figure 5**.

Once all the days have been split into separate files and written out (see step 7 in **Figure 2**), I simply move the file to the processed-

Figure 5 Selecting Each Day's Data

```
// Loop through collection filtering by day
for(int dayToProcess = 1; dayToProcess <= maxDayOfMonth; dayToProcess++)
{
    // Get documents for current processing day
    var docsForCurrentDay = results.Where (d => d.Day==dayToProcess);

    // Serialize to string
    string jsonForCurrentDay =
        JsonConvert.SerializeObject(docsForCurrentDay);
    log.Info($"***** Docs for day {dayToProcess} *****");

    // Get date for one of the records for today
    string processDateString = (from docs in results where docs.Day ==
        dayToProcess select docs.Date).First();

    path = "newdailysplit/" + DateTime.Parse(processDateString).ToString("yyyyMMdd")
        + "-" + enrollment + "-dailysplit.json";

    // Write out each day's data to file in container "newdailysplit"
    var attributes = new Attribute[]
    {
        new BlobAttribute(path),
        new StorageAccountAttribute("eabillingstorage_STORAGE")
    };

    using (var writer = await binder.BindAsync<TextWriter>(attributes))
    {
        writer.Write(jsonForCurrentDay);
    }
}
```



We Are Changing the Way You Look at Reporting

A Reporting Q&A with Bjoern Meyer, Text Control

Founded in 1991, Text Control is an award-winning Visual Studio Industry Partner and leading vendor of word processing and reporting components for Windows, web and mobile development technologies.

Q What is Text Control doing?

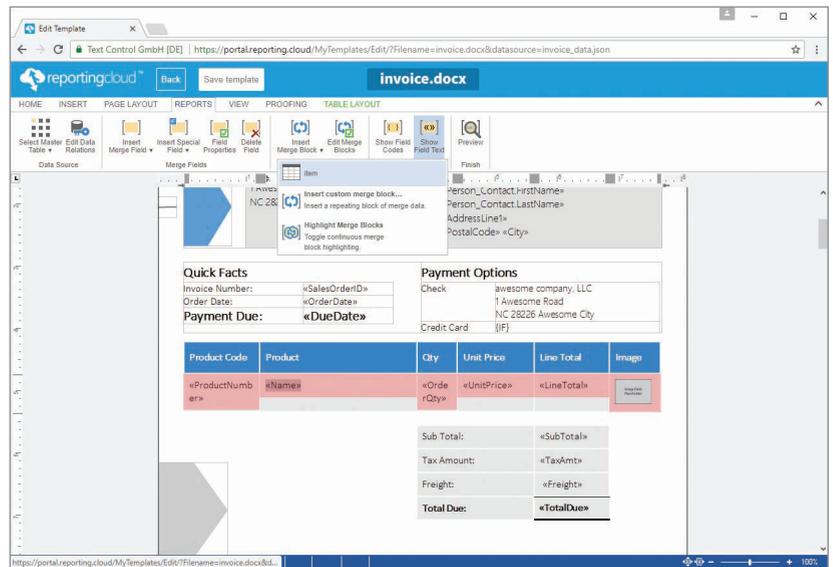
A Since 25 years, our products and technologies help thousands of developers add comprehensive reporting and word processing functionality to their applications. Our mission is to use everyday innovation to uncover our user's real reporting requirements.

Digital transformation changed every process in today's business world. The number of e-commerce transactions skyrocket and supply chains are fully connected. In nearly any business process, documents and reports need to be designed, created, shared and archived. Our technologies help companies to integrate document processing to client, web and cloud solutions to gain the largest competitive advantage.

We have been developing software components for reporting and document processing for more than 25 years. We are continually looking for new and innovative ways to improve document processing to make these processes easier for end-users and more efficient.

Q What is the Text Control Reporting Framework?

A The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor. Users can create documents and templates using ordinary Microsoft Word skills. It is completely independent from MS Word or any other third-party application and can be completely integrated into business applications. The Text Control Reporting Framework is included in all .NET based TX Text Control products including ASP.NET, Windows Forms and WPF.



Q What sets Text Control Reporting apart from other reporting vendors?

A Text Control Reporting is based on the powerful word processing component TX Text Control. The MS Word compatible template can be merged with a data object (business object) or database content with one line of code. At the same time, Text Control provides a powerful API to customize this merge process completely. The report generation can be fully integrated into .NET applications.

Q Tell us more about your new Cloud reporting Web API

A Text Control ReportingCloud brings complete reporting functionality to the cloud so all developers can use it, irrespective of the platform or language they're using. Its highly RESTful API can be used to merge Microsoft Word compatible templates with JSON data from all clients including .NET, JavaScript, PHP, Node.JS, jQuery, Ruby, Python, Android, Java and iOS.

For more information, visit →

www.textcontrol.com

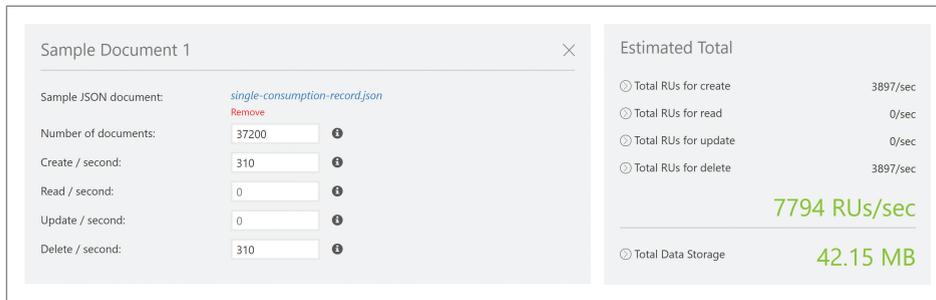


Figure 6 Azure Cosmos DB Pricing Calculator

usage container. To keep the diagram in **Figure 2** easy to parse, I've omitted some containers—in particular, the error files container is missing from the diagram. This is the container that holds any file that causes an exception during processing, whether that file is the entire usage file or just one of the daily splits. I don't spend time or effort correcting the data for missing or errored days because, once an issue is identified, the process can be triggered for a given month and enrollment or for a single daily split to correct the problem. Also clearly missing from the prototype are alerting and compensating mechanisms for when errors occur, but that's something I want to bubble up through Application Insights integration with the Operations Management Suite.

Persisting the Data to Azure Cosmos DB

With the files split and ready to be picked up by the ProcessDaily-Usage Function, it's time to consider some issues that need to be addressed, namely throughput to the target and how to handle updates. Often when working through some solution architecture in an enterprise, you run into older systems that are less capable, or where real-time loads and high-throughput scenarios need to be managed. I don't naturally have any hard throughput constraints in my cloud native setup for this architecture, but I could create problems for myself if I don't take the time to think through the volume and speed of the data I'm feeding into the cloud services I'm consuming.

For my data set, each of the daily splits is about 2.4MB and contains about 1,200 individual documents. Keep in mind that each document represents one meter reading for one resource provisioned in Azure. Thus, for each EA the number of documents in a daily split could vary greatly depending on resource usage across the enterprise. The ProcessDailyUsage Function is configured to trigger based on receiving new blobs in the newdailyssplit container. This means I'll have as many as 31 concurrent Function executions manipulating the data. To help me estimate what I need to provision for Azure Cosmos DB, I used the calculator at documentdb.com/capacityplanner. Without some empirical testing I had to make a few guesses for the

following as my rules for estimating:

- 1,200 records
- 31 concurrent executions (for a single EA)
- 0.124 seconds per request (empirical evidence from measuring a few individual requests)

I'll round down to 0.1 seconds for a more conservative estimate, thus overestimating the load. This nets 310 requests per second per EA, which in turn comes out to about 7,800 request units (RUs) based on the calculator results, as can be seen in **Figure 6**.

Because the maximum RUs that can be provisioned without calling support is 10,000, this might seem kind of high. However, I'm running an unthrottled parallel process and that drives up the throughput significantly, which in turn will drive up the cost. This is a major consideration when designing the structure because it's fine for me to run this for some testing, but for the real solution I'll need a throttling mechanism to slow down the processing so I can provision fewer RUs and save myself a little money. I don't need the data to be captured as fast as possible, just within a reasonable enough time that someone could review and consume it on a daily basis. The good news is that the Azure Functions team

has a concurrency control mechanism in the backlog of issues that will eventually get resolved (bit.ly/2tcpAb1), and will provide a good means of control once implemented. Some other options are to introduce artificial arbitrary delays (let's all agree this is bad) or to rework the processing and handle the parallel execution explicitly in the C# code. Also, as technical expert Fabio Cavalcante pointed out in a conversation, another good option would be to modify the architecture a bit by adding Azure Storage Queues and using features such as visibility timeouts and scheduled delivery to act as a throttling mechanism. That would add a few moving parts to the system and I'd have to work out the interaction of using a queue for activation while keeping the data in storage, or slice up the data in 64KB blocks for the queue. Once throttling is available in Azure Functions, I'll be able to keep it in this simpler form with which I'm working. The salient point here is that when working with a serverless architecture you must be familiar with the

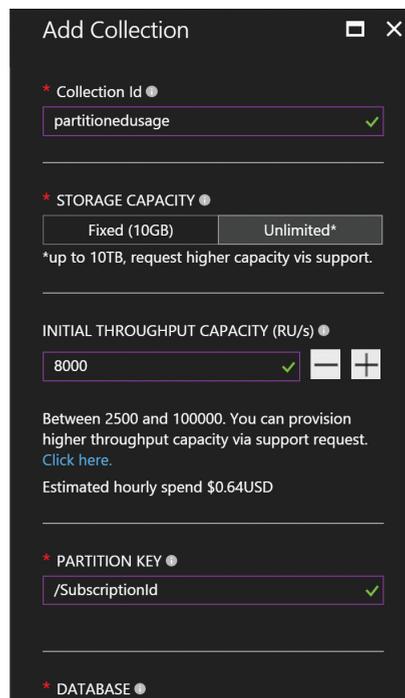


Figure 7 Provisioning a New Collection



RD
—

Rider

New .NET IDE

**Cross-platform.
Killer code analysis.
Great for refactoring.**

From the makers of ReSharper,
IntelliJ IDEA, and WebStorm.

Learn more
and download
jetbrains.com/rider



JET
BRAINS
—

Figure 8 Using FeedOptions to Set the Cross-Partition Query Flag

```
string docsToDeleteQuery = String.Format(@"SELECT * FROM c where c.Enrollment =
    ""{0}"" AND c.Date = ""{1}""", enrollment, incomingDataDate);

FeedOptions queryOptions = new FeedOptions { MaxItemCount = -1,
    EnableCrossPartitionQuery = true };

IQueryable<Document> deleteQuery = docDBClient.
CreateDocumentQuery<Document>(
    UriFactory.CreateDocumentCollectionUri(dbName, collectionName),
    new SqlQuerySpec(docsToDeleteQuery), queryOptions);

log.Info("Delete documents");
int deletedDocumentCount = 0;
foreach (Document doc in deleteQuery)
{
    await docDBClient.DeleteDocumentAsync(((dynamic)doc)._self,
        new RequestOptions { PartitionKey =
            new PartitionKey(((dynamic)doc).SubscriptionId) });
    deletedDocumentCount++;
}
```

constraints of the platforms on which you're building, as well as the cost of each decision.

When provisioning more than 2,500 RUs, the system requires that a partition key be specified. This works for me, because I want to partition that data in any case to help with both scale and security in the future.

As you can see in **Figure 7**, I've specified 8,000 RUs, which is a little more than the calculation indicated, and I've specified SubscriptionId as the partition key.

Additionally, I set up the ProcessDailyUsage with a blob trigger on the newdailySplit container and with an input and output binding for Azure Cosmos DB. The input binding is used to find the records that exist for the given day and enrollment and to handle duplicates. I'll ensure that my FeedOptions sets the cross-partition query flag, as shown in **Figure 8**.

I create a query to grab all the records for the enrollment on that date and then loop through and delete them. This is one instance where SQL Azure could've made things easier by issuing a DELETE query or by using an upsert with a known primary key. However, in Azure Cosmos DB, to do the upsert I need the row ID, which means I must make the round trip and do the comparison on fields I know to uniquely identify the document and then use that row's id or selflink. For this example, I simply delete all the

Figure 9 The Log Information from a Daily Split File

```
2017-06-10T01:16:55.291 Function started (Id=bfb220aa-97ab-4d36-9c1e-602763b93ff0)
2017-06-10T01:16:56.041 First 15 chars: [{"AccountOwner
2017-06-10T01:16:56.181 get date
2017-06-10T01:16:56.181 getting enrollment
2017-06-10T01:16:56.181 Incoming date: 11/01/2016 for Enrollment: 4944727
2017-06-10T01:16:56.181 Collection: partitionedusage
2017-06-10T01:16:56.181 query: SELECT * FROM c where c.Enrollment =
    "4944727" AND c.Date = "11/01/2016"
2017-06-10T01:16:56.181 Create delete query
2017-06-10T01:16:56.197 Delete documents
2017-06-10T01:17:23.189 2142 docs deleted while processing
20161101-4944727-dailysplit.json
2017-06-10T01:17:23.189 Import documents
2017-06-10T01:17:44.628 2142 records imported from file
20161101-4944727-dailysplit.json
2017-06-10T01:17:44.628 Moving file 20161101-4944727-dailysplit.json to /
    processedusage container
2017-06-10T01:17:44.674 Deleting 20161101-4944727-dailysplit.json
2017-06-10T01:17:44.690 Completed!
2017-06-10T01:17:44.690 Function completed (Success, Id=bfb220aa-97ab-
    4d36-9c1e-602763b93ff0, Duration=49397ms)
```

records and then add the new—and potentially updated—documents back in. To do this I need to pass in the partition key to the DeleteDocumentAsync method. An optimization would be to pull the documents back and do a local comparison, update any changed documents and add net new documents. It's a little taxing, because all of the elements in each document must be compared. Because there's no primary key defined for the billing documents, you can likely find the matched document using SubscriptionId, MeterId, InstanceId and Date and compare the rest of the elements from there. This would offload some of the work from Azure Cosmos DB and reduce the overall traffic.

With the way cleared to add the documents back into the collection, I simply loop through the docs and call AddAsync on the documentCollector I defined as the output binding for the Azure Function:

```
// Update the enrollment field in the incoming collection
incomingDailyUsage.ForEach (usage => usage.Enrollment = enrollment);

int processedRecordCount=0;
foreach (EnrollmentUsageDetail usageDoc in incomingDailyUsage)
{
    await documentCollector.AddAsync(usageDoc);
    processedRecordCount++;
}
```

While it's not much of a change, I've also done a little bit of enrichment by adding the Enrollment number to each document in the collection. Running one daily split file produces the log information shown in **Figure 9**.

Final Note

The only thing left to do is to run a good many iterations with varying inputs and then measure so I can properly size the services I'm using. This includes testing out the geographic replication capabilities and some further prototyping of the security that I'll want to implement around subscription data access; these were two of the major reasons for choosing Azure Cosmos DB. The net lessons to be gleaned are some of the ones that we seem to keep learning in the world of IT:

1. There are no magic bullets, not even with a serverless architecture.
2. Nothing replaces thorough testing.
3. Size your dependent services and treat this as seriously as you did when sizing your hardware in the past.
4. Pay close attention to cost, especially under high throughput conditions.

The upside of using serverless compute like Azure Functions is that you pay only for what's consumed. For regular but infrequent processing such as this, that can be a big benefit in cost savings. Finally, configuring capabilities is a better experience and allows faster time to product than configuring host servers. ■

JOSEPH FULTZ is a cloud solution architect at Microsoft. He works with Microsoft customers developing architectures for solving business problems leveraging Microsoft Azure. Formerly, Fultz was responsible for the development and architecture of GM's car-sharing program (mavendrive.com). Contact him on Twitter: @JosephRFultz or via e-mail at jofultz@microsoft.com.

THANKS to the following Microsoft technical expert who reviewed this article: Fabio Calvacante

ROCK YOUR CODE TOUR 2017



REDMOND
AUG
14-18

Redmond

August 14-18

See pages 70-71



CHICAGO
SEPT
18-21

Chicago

September 18-21

See pages 54-55



ANAHEIM
OCT
16-19

Anaheim

October 16-19

See page 56-57



ORLANDO
NOV
12-17

Orlando

November 12-17

See pages 76-79





INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics Include:

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Native Client
- Software Practices
- Database and Analytics
- Angular JS
- ASP.NET / Web Server
- Agile
- ALM / DevOps
- Cloud Computing
- Windows Client
- Xamarin



Register By August 18 and Save \$200!

Use promo code VSLCH5

**REGISTER
NOW**

GOLD SPONSOR



SUPPORTED BY



Visual Studio



Visual Studio
MAGAZINE

PRODUCED BY



CHICAGO AGENDA AT-A-GLANCE

ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server
--------------	-----------------	------------------------	---------------	--------------------	--------------------------------	------------	------------

START TIME END TIME Visual Studio Live! Pre-Conference Workshops: Monday, September 18, 2017 (Separate entry fee required)

7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries					
9:00 AM	6:00 PM	M01 Workshop: Distributed Cross-Platform Application Architecture - Jason Bock & Rockford Lhotka	M02 Workshop: Practical ASP.NET DevOps with VSTS or TFS - Brian Randell	M03 Workshop: SQL Server 2016 for Developers - Andrew Brust & Leonard Lobel			
6:45 PM	9:00 PM	Dine-A-Round					

START TIME END TIME Visual Studio Live! Day 1: Tuesday, September 19, 2017

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 JavaScript for the C# Developer - Philip Japikse	T02 Storyboarding 101 - Billy Hollis	T03 Build Cross-Platform Apps in C# using CSLA.NET - Rockford Lhotka	T04 What's New in Visual Studio 2017 - Robert Green		
10:45 AM	12:00 PM	T05 TypeScript: The Future of Front End Web Development - Ben Hoelting	T06 Better, Faster, Automated! Windows App Deployment in Visual Studio Mobile Center - Ela Malani & Piyush Joshi	T07 Roll Your Own Dashboard in XAML - Billy Hollis	T08 What's New in C#7 - Jason Bock		
12:00 PM	1:00 PM	Lunch - Visit Exhibitors					
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors					
1:30 PM	2:45 PM	T09 ASP.NET Core MVC - What You Need to Know - Philip Japikse	T10 Professional Scrum Development Using Visual Studio 2017 - Richard Hundhausen	T11 What's New for Developers in SQL Server 2016 - Leonard Lobel	T12 To Be Announced		
3:00 PM	4:15 PM	T13 User Authentication for ASP.NET Core MVC Applications - Brock Allen	T14 PowerShell for Developers - Brian Randell	T15 What's New in Azure IaaS v2 - Eric D. Boyd	T16 To Be Announced		
4:15 PM	5:30 PM	Welcome Reception					

START TIME END TIME Visual Studio Live! Day 2: Wednesday, September 20, 2017

7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	W01 Securing Web APIs in ASP.NET Core - Brock Allen	W02 Professional Software Testing Using Visual Studio 2017 - Richard Hundhausen	W03 Cloud Oriented Programming - Vishwas Lele	W04 Database Development with SQL Server Data Tools - Leonard Lobel		
9:30 AM	10:45 AM	W05 Assembling the Web - A Tour of WebAssembly - Jason Bock	W06 Get Started with Git and GitHub - Robert Green	W07 Building Modern Web Apps with Azure - Eric D. Boyd	W08 Tactical DevOps for SQL Server - Brian Randell		
11:00 AM	12:00 PM	General Session: To Be Announced					
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch - Visit Exhibitors					
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	W09 Tools for Modern Web Development Dev Ops - Ben Hoelting	W10 Go Mobile with C#, Visual Studio, and Xamarin - James Montemagno	W11 Build Awesome AF Apps! - Rachel Appel	W12 Power BI: Analytics for Desktop, Mobile and Cloud - Andrew Brust		
3:00 PM	4:15 PM	W13 Get Rid of HTML Tables for Better Mobile Web Applications - Paul Sheriff	W14 Continuous Integration & Deployment for Mobile Apps - James Montemagno	W15 Microservices with Azure Container Service & Service Fabric - Vishwas Lele	W16 Power BI: Beyond the Basics - Andrew Brust		
4:30 PM	5:45 PM	W17 Use HTML5/Bootstrap to Build Business UI - Paul Sheriff	W18 Mobilizing your Existing Enterprise Applications - Nick Landry	W19 Busy Developer's Guide to the Google Cloud Platform - Ted Neward	W20 Big Data Solutions in Azure - David Giard		
7:00 PM	9:00 PM	Visual Studio Live! Evening Event					

START TIME END TIME Visual Studio Live! Day 3: Thursday, September 21, 2017

7:30 AM	8:00 AM	Web Client					
8:00 AM	9:15 AM	TH01 I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(Me), Maybe? - Rachel Appel	TH02 PowerApps, Flow, and Common Data Service: Empowering Businesses with the Microsoft Business Application Platform - Archana Nair	TH03 Lessons Learned from Real World Xamarin.Forms Projects - Nick Landry	TH04 Busy Developer's Guide to NoSQL - Ted Neward		
9:30 AM	10:45 AM	TH05 Build Object-Oriented Enterprise Apps in JavaScript with TypeScript - Rachel Appel	TH06 PowerApps and Flow Part II: Package, Embed, and Extend Your Applications - Anousha Mesbah & Pratap Ladhani	TH07 Improve Your Retrospective Outcomes with Agile Kaizen - Angela Dugan	TH08 Building Applications with DocumentDb - New Features and Best Practices - Raj Krishnan		
11:00 AM	12:15 PM	TH09 What's New in TypeScript? - Doris Chen	TH10 Getting Started with Entity Framework Core - Jim Wooley	TH11 Open Source for Microsoft Developers - Rockford Lhotka	TH12 Introduction to Machine Learning with R - Raj Krishnan		
12:15 PM	1:15 PM	Lunch					
1:15 PM	2:30 PM	TH13 Practical Performance Tips and Tricks to Make Your HTML/JavaScript Faster - Doris Chen	TH14 Getting Your Agile Team Unstuck! Tips and Tricks for Blasting Through Common Setbacks - Angela Dugan	TH15 DI Why? Getting a Grip on Dependency Injection - Jeremy Clark	TH16 The Rise of the Machines - Machine Learning for Developers - Adam Tuliper		
2:45 PM	4:00 PM	TH17 Building Powerful Applications with AngularJS 2 and TypeScript - David Giard	TH18 Improving Code Quality with Roslyn Code Analyzers - Jim Wooley	TH19 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - Jeremy Clark	TH20 I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - Adam Tuliper		

Speakers and sessions subject to change

CONNECT WITH US



vslive.com/chicagomsdn

Visual Studio **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

ANAHEIM, CA
OCT 16-19, 2017
HYATT REGENCY
A Disneyland® Good Neighbor Hotel

California

CODIN!

**INTENSE TRAINING FOR DEVELOPERS, ENGINEERS,
PROGRAMMERS, ARCHITECTS AND MORE!**

Development Topics include:

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing



Register by August 25 and Save \$300!

Use promo code VSLAN5

**REGISTER
NOW**

EVENT PARTNER



SUPPORTED BY



Visual Studio



Visual Studio
MAGAZINE

PRODUCED BY



ANAHEIM AGENDA AT-A-GLANCE

ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server
--------------	-----------------	------------------------	---------------	--------------------	--------------------------------	------------	------------

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, October 16, 2017 <small>(Separate entry fee required)</small>					
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries					
9:00 AM	6:00 PM	M01 Workshop: Distributed Cross-Platform Application Architecture - Jason Bock & Rockford Lhotka	M02 Workshop: Practical ASP.NET DevOps with VSTS or TFS - Brian Randell	M03 Workshop: Developer Dive into SQL Server 2016 - Leonard Lobel			
6:45 PM	9:00 PM	Dine-A-Round					
START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, October 17, 2017					
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 What's New in TypeScript? - Doris Chen	T02 Go Mobile with C#, Visual Studio, and Xamarin - James Montemagno	T03 To Be Announced		T04 What's New in Visual Studio 2017 - Robert Green	
10:45 AM	12:00 PM	T05 Build Object-Oriented Enterprise Apps in JavaScript with TypeScript - Rachel Appel	T06 Optimizing and Extending Xamarin.Forms Mobile Apps - James Montemagno	T07 What's New for Developers in SQL Server 2016 - Leonard Lobel		T08 To Be Announced	
12:00 PM	1:30 PM	Lunch					
1:30 PM	2:45 PM	T09 Angular 101: Part 1 - Deborah Kurata	T10 Get Started with Git and GitHub - Robert Green	T11 Exploring T-SQL Enhancements: Windowing and More - Leonard Lobel		T12 Microsoft Teams - More Than Just Chat! - Nedra Allmond	
3:00 PM	4:15 PM	T13 Angular 101: Part 2 - Deborah Kurata	T14 Do It Again, Faster! Automate Your Windows Deployment Pipeline - Ela Malani	T15 What's New in Azure IaaS v2 - Eric D. Boyd		T16 Open Source for the Microsoft Developer - Rockford Lhotka	
4:15 PM	5:30 PM	Welcome Reception					
START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, October 18, 2017					
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	W01 I Just Met You, and "This" is Crazy, But Here's My NaN, So Call (Me), Maybe? - Rachel Appel	W02 Tactical DevOps with VSTS - Brian Randell	W03 Go Serverless with Azure Functions - Eric D. Boyd		W04 What's New in C#7 - Jason Bock	
9:30 AM	10:45 AM	W05 Practical Performance Tips and Tricks to Make Your HTML/JavaScript Faster - Doris Chen	W06 Real World VSTS Usage for the Enterprise - Jim Szubryt	W07 Cloud Oriented Programming - Vishwas Lele		W08 I'll Get Back to You: Understanding Task, Await, and Asynchronous Methods - Jeremy Clark	
11:00 AM	12:00 PM	General Session: To Be Announced					
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch					
1:30 PM	2:45 PM	W09 Assembling the Web - A Tour of WebAssembly - Jason Bock	W10 Database Lifecycle Management and the SQL Server Database - Brian Randell	W11 Microservices with Azure Container Service & Service Fabric - Vishwas Lele		W12 Getting Started with Entity Framework Core - Jim Wooley	
3:00 PM	4:15 PM	W13 Building Single Page Web Applications Using Aurelia.js and the MVVM Pattern - Ben Hoelting	W14 Getting to SAFE in the Enterprise - Jim Szubryt	W15 Busy Developer's Guide to the Clouds - Ted Neward		W16 Improving Code Quality with Roslyn Code Analyzers - Jim Wooley	
4:30 PM	05:45 PM	W17 Securing Angular Apps - Brian Noyes	W17 Building Applications with DocumentDb - New Features and Best Practices - Raj Krishnan	W18 Busy Developer's Guide to the Google Cloud Platform - Ted Neward		W19 Learn to Love Lambdas (and LINQ, Too) - Jeremy Clark	
7:00 PM	9:00 PM	Visual Studio Live! Evening Event					
START TIME	END TIME	Visual Studio Live! Day 3: Thursday, October 19, 2017					
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	TH01 ASP.NET Core MVC - What You Need to Know - Philip Japikse	TH02 Tools for Modern Web Development Dev Ops - Ben Hoelting	TH03 The Rise of the Machines - Machine Learning for developers - Adam Tuliper		TH04 Storyboarding 101 - Billy Hollis	
9:30 AM	10:45 AM	TH05 Role-Based Security Stinks: How to Implement Better Authorization in ASP.NET and ASP.NET Core - Benjamin Day	TH06 Building Cross-Platform Apps in C# using CSLA .NET - Rockford Lhotka	TH07 Introduction to Machine Learning with R - Raj Krishnan		TH08 Agile: You Keep Using That Word... - Philip Japikse	
11:00 AM	12:15 PM	TH09 From Zero to the Web API - Paul Sheriff	TH10 Programming with the Model-View-ViewModel Pattern - Miguel Castro	TH11 I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - Adam Tuliper		TH12 Top 10 Ways to Go from Good to Great Scrum Master - Benjamin Day	
12:15 PM	1:15 PM	Lunch					
1:15 PM	2:30 PM	TH13 Cortana Everywhere: Speech, Conversation & Skills Development - Nick Landry	TH14 Roll Your Own Dashboard in XAML - Billy Hollis	TH15 Unit Testing T-SQL Code - Steve Jones		TH16 Exposing an Extensibility API for your Applications - Miguel Castro	
2:45 PM	4:00 PM	TH17 Securing Web Apps and APIs with IdentityServer - Brian Noyes	TH18 Windows 10 for Developers: Building Universal Apps for 1B Devices - Nick Landry	TH19 A Tour of SQL Server Security Features - Steve Jones		TH20 Real World Applications for Dependency Injection - Paul Sheriff	

Speakers and sessions subject to change

CONNECT WITH US



vslive.com/anaheimmsdn



Deep Neural Network IO Using C#

Many of the recent advances in machine learning (making predictions using data) have been realized using deep neural networks. Examples include speech recognition in Microsoft Cortana and Apple Siri, and the image recognition that helps enable self-driving automobiles.

The term deep neural network (DNN) is general and there are several specific variations, including recurrent neural networks (RNNs) and convolutional neural networks (CNNs). The most basic form of a DNN, which I explain in this article, doesn't have a special name, so I'll refer to it just as a DNN.

This article will introduce you to DNNs so you'll have a concrete demo program to experiment with, which will help you to understand literature on DNNs. I won't present code that can be used directly in a production system, but the code can be extended to create such a system, as I'll explain. Even if you never intend to implement a DNN, you might find the explanation of how they work interesting for its own sake.

A DNN is best explained visually. Take a look at **Figure 1**. The deep network has two input nodes, on the left, with values (1.0, 2.0). There are three output nodes on the right, with values (0.3269, 0.3333, 0.3398). You can think of a DNN as a complex math function that typically accepts two or more numeric input values and returns one or more numeric output values.

The DNN shown might correspond to a problem where the goal is to predict the political party affiliation (Democrat, Republican, Other) of a person based on age and income, where the input values are scaled in some way. If Democrat is encoded as (1,0,0) and Republican is encoded as (0,1,0) and Other is encoded as (0,0,1), then the DNN in **Figure 1** predicts Other for someone with age = 1.0 and income = 2.0 because the last output value (0.3398) is the largest.

A regular neural network has a single hidden layer of processing nodes. A DNN has two or more hidden layers and can handle very difficult prediction problems. Specialized types of DNNs, such as RNNs and CNNs, also have multiple layers of processing nodes, but more complicated connection architectures, as well.

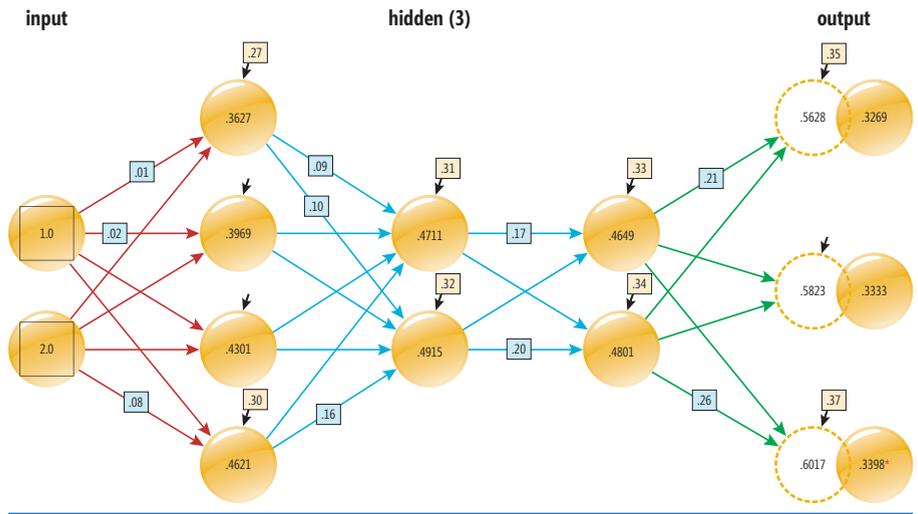


Figure 1 A Basic Deep Neural Network

The DNN in **Figure 1** has three hidden layers of processing nodes. The first hidden layer has four nodes, the second and third hidden layers have two nodes. Each long arrow pointing from left to right represents a numeric constant called a weight. If nodes are zero-base indexed with [0] at the top of the figure, then the weight connecting input[0] to hidden[0][0] (layer 0, node 0) has value 0.01 and the weight connecting input[1] to hidden[0][3] (layer 0, node 3) has value 0.08 and so on. There are 26 node-node weight values.

Each of the eight hidden and three output nodes has a small arrow that represents a numeric constant called a bias. For example, hidden[2][0] has bias value of 0.33 and output[1] has a bias value of 0.36. Not all of the weights and bias values are labeled in the diagram, but because the values are sequential between 0.01 and 0.37, you can easily determine the value of a non-labeled weight or bias.

In the sections that follow, I explain how the DNN input-output mechanism works and show how to implement it. The demo program is coded using C#, but you shouldn't have too much trouble refactoring the code to another language, such as Python or JavaScript, if you wish to do so. The demo program is too long to present in its entirety in this article, but the complete program is available in the accompanying code download.

The Demo Program

A good way to see where this article is headed is to examine the screenshot of the demo program in **Figure 2**. The demo corresponds to the

Code download available at msdn.com/magazine/0717magcode.

Build More

with GrapeCity Developer Solutions

Visual Studio-Integrated UI Controls and Developer Productivity Tools for Delivering Enterprise Apps Across All Platforms and Devices



ComponentOne Studio

Elegant, modular .NET UI controls for Visual Studio



Visual Studio

Powerful development tools engineered for Visual Studio developers

ar

ActiveReports

Powerful .NET reporting platform for essential business needs

sp

Spread Studio

Versatile .NET spreadsheet data and UI components



ComponentOne Studio for Xamarin

Cross-platform grids, charts, and UI controls for native mobile devices



Wijmo JavaScript

Fast, lightweight true JavaScript controls written in TypeScript



GrapeCity's family of products provides **developers, designers,** and **architects** with the ultimate collection of easy-to-use tools for building **sleek, high-performing, feature-complete** applications. With over 25 years of experience, we understand your needs and offer the industry's best support. **Our team is your team.** For more information: **1.800.858.2739**

Learn more and get free 30-day trials at tools.grapecity.com

```

file:///C:/DeepNetInputOutput/bin/Debug/DeepNetInputOutp...
Begin deep net IO demo
Creating a 2-(4-2-2)-3 deep neural network
Setting weights and biases to 0.01 to 0.37
Computing output values for input = [1.0, 2.0]

input node [0] = 1.0000
input node [1] = 2.0000

hidden layer 0 node [0] = 0.3627
hidden layer 0 node [1] = 0.3969
hidden layer 0 node [2] = 0.4301
hidden layer 0 node [3] = 0.4621

hidden layer 1 node [0] = 0.4711
hidden layer 1 node [1] = 0.4915

hidden layer 2 node [0] = 0.4649
hidden layer 2 node [1] = 0.4801

output node [0] = 0.3269
output node [1] = 0.3333
output node [2] = 0.3398

End demo

```

Figure 2 Basic Deep Neural Network Demo Run

DNN shown in **Figure 1** and illustrates the input-output mechanism by displaying the values of the 13 nodes in the network. The demo code that generated the output begins with the code shown in **Figure 3**.

Notice that the demo program uses only plain C# with no namespaces except for System. The DNN is created by passing the number of nodes in each layer to a DeepNet program-defined class constructor. The number of hidden layers, 3, is passed implicitly as the number of items in the numHidden array. An alternative design is to pass the number of hidden layers explicitly.

The values of the 26 weights and the 11 biases are set like so:

```

int nw = DeepNet.NumWeights(numInput, numHidden, numOutput);
Console.WriteLine("Setting weights and biases to 0.01 to " +
(nw/100.0).ToString("F2"));
double[] wts = new double[nw];
for (int i = 0; i < wts.Length; ++i)
    wts[i] = (i + 1) * 0.01;
dn.SetWeights(wts);

```

The total number of weights and biases is calculated using a static class method NumWeights. If you refer back to **Figure 1**, you can see that because each node is connected to all nodes in the layer to the right, the number of weights is $(2*4) + (4*2) + (2*2) + (2*3) = 8 + 8 + 4 + 6 = 26$. Because there's one bias for each hidden and output node, the total number of biases is $4 + 2 + 2 + 3 = 11$.

Figure 3 Beginning of Output-Generating Code

```

using System;
namespace DeepNetInputOutput
{
    class DeepInputOutputProgram
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Begin deep net IO demo");
            Console.WriteLine("Creating a 2-(4-2-2)-3 deep network");
            int numInput = 2;
            int[] numHidden = new int[] { 4, 2, 2 };
            int numOutput = 3;
            DeepNet dn = new DeepNet(numInput, numHidden, numOutput);
            ...
        }
    }
}

```

An array named wts is instantiated with 37 cells and then the values are set to 0.01 through 0.37. These values are inserted into the DeepNet object using the SetWeights method. In a realistic, non-demo DNN, the values of the weights and biases would be determined using a set of data that has known input values and known, correct output values. This is called training the network. The most common training algorithm is called back-propagation.

The Main method of the demo program concludes with:

```

...
Console.WriteLine("Computing output for [1.0, 2.0] ");
double[] xValues = new double[] { 1.0, 2.0 };
dn.ComputeOutputs(xValues);
dn.Dump(false);
Console.WriteLine("End demo");
Console.ReadLine();
} // Main
} // Class Program

```

Method ComputeOutputs accepts an array of input values and then uses the input-output mechanism, which I'll explain shortly, to calculate and store the values of the output nodes. The Dump helper method displays the values of the 13 nodes, and the "false" argument means to not display the values of the 37 weights and biases.

The Input-Output Mechanism

The input-output mechanism for a DNN is best explained with a concrete example. The first step is to use the values in the input nodes to calculate the values of the nodes in the first hidden layer. The value of the top-most hidden node in the first hidden layer is:

$$\tanh((1.0)(0.01) + (2.0)(0.05) + 0.27) = \tanh(0.38) = 0.3627$$

In words, "compute the sum of the products of each input node and its associated weight, add the bias value, then take the hyperbolic tangent of the sum." The hyperbolic tangent, abbreviated tanh, is called the activation function. The tanh function accepts any value from negative infinity to positive infinity, and returns a value between -1.0 and +1.0. Important alternative activation functions include the logistic sigmoid and rectified linear (ReLU) functions, which are outside the scope of this article.

The values of the nodes in the remaining hidden layers are calculated in exactly the same way. For example, hidden[1][0] is:

$$\tanh((0.3627)(0.09) + (0.3969)(0.11) + (0.4301)(0.13) + (0.4621)(0.15) + 0.31) = \tanh(0.5115) = 0.4711$$

And hidden[2][0] is:

$$\tanh((0.4711)(0.17) + (0.4915)(0.19) + 0.33) = \tanh(0.5035) = 0.4649$$

The values of the output nodes are calculated using a different activation function, called softmax. The preliminary, pre-activation sum-of-products plus bias step is the same:

$$\text{pre-activation output}[0] = (.4649)(0.21) + (0.4801)(0.24) + 0.35 = 0.5628$$

$$\text{pre-activation output}[1] = (.4649)(0.22) + (0.4801)(0.25) + 0.36 = 0.5823$$

$$\text{pre-activation output}[2] = (.4649)(0.23) + (0.4801)(0.26) + 0.37 = 0.6017$$

HTML5 Viewer & Document Management Kit

NEW RELEASE



Easy integration



Full support for custom
snap-in



Zero-footprint solution



Fully customizable UI



Mobile devices
optimization



Fast & crystal-clear
rendering

Check the **New Features** and the **Online Demos**

**DOWNLOAD
YOUR FREE TRIAL**

www.docuvieware.com

The softmax of three arbitrary values, x, y, z is:

$$\begin{aligned} \text{softmax}(x) &= e^x / (e^x + e^y + e^z) \\ \text{softmax}(y) &= e^y / (e^x + e^y + e^z) \\ \text{softmax}(z) &= e^z / (e^x + e^y + e^z) \end{aligned}$$

where e is Euler's number, approximately 2.71828. So, for the DNN in **Figure 1**, the final output values are:

$$\begin{aligned} \text{output}[0] &= e^{0.5628} / (e^{0.5628} + e^{0.5823} + e^{0.6017}) = 0.3269 \\ \text{output}[1] &= e^{0.5823} / (e^{0.5628} + e^{0.5823} + e^{0.6017}) = 0.3333 \\ \text{output}[2] &= e^{0.6017} / (e^{0.5628} + e^{0.5823} + e^{0.6017}) = 0.3398 \end{aligned}$$

The purpose of the softmax activation function is to coerce the output values to sum to 1.0 so that they can be interpreted as probabilities and map to a categorical value. In this example, because the third output value is the largest, whatever categorical value that was encoded as (0,0,1) would be the predicted category for inputs = (1.0, 2.0).

Implementing a DeepNet Class

To create the demo program, I launched Visual Studio and selected the C# Console Application template and named it DeepNetInput-Output. I used Visual Studio 2015, but the demo has no significant .NET dependencies, so any version of Visual Studio will work.

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

I implemented the demo DNN as a class named DeepNet. The class definition begins with:

```
public class DeepNet
{
    public static Random rnd;
    public int nInput;
    public int[] nHidden;
    public int nOutput;
    public int nLayers;
    ...
}
```

All class members are declared with public scope for simplicity. The static Random object member named rnd is used by the DeepNet class to initialize weights and biases to small random values (which are then overwritten with values 0.01 to 0.37). Members nInput and nOutput are the number of input and output nodes. Array member nHidden holds the number of nodes in each hidden layer, so the number of hidden layers is given by the Length property of the array, which is stored into member nLayers for convenience. The class definition continues:

```
public double[] iNodes;
public double [][] hNodes;
public double[] oNodes;
```

A deep neural network implementation has many design choices. Array members iNodes and oNodes hold the input and output values, as you'd expect. Array-of-arrays member hNodes holds the hidden node values. An alternative design is to store all nodes in a single array-of-arrays structure nnNodes, where in the demo nnNodes[0] is an array of input node values and nnNodes[4] is an array of output node values.

The node-to-node weights are stored using these data structures:

```
public double[][] ihWeights;
public double[][][] hhWeights;
public double[][] hoWeights;
```

Member ihWeights is an array-of-arrays-style matrix that holds the input-to-first-hidden-layer weights. Member hoWeights is an array-of-arrays-style matrix that holds the weights connecting the last hidden layer nodes to the output nodes. Member hhWeights is an array where each cell points to an array-of-arrays matrix that holds the hidden-to-hidden weights. For example, hhWeights[0][3][1] holds the weights connecting hidden node [3] in hidden layer [0] to hidden node [1] in hidden layer [0+1]. These data structures are the heart of the DNN input-output mechanism and are a bit tricky. A conceptual diagram of them is shown in **Figure 4**.

The last two class members hold the hidden node biases and the output node biases:

```
public double[][] hBiases;
public double[] oBiases;
```

As much as any software system I work with, DNNs have many alternative data structure designs, and having a sketch of these data structures is essential when writing input-output code.

Computing the Number of Weights and Biases

To set the weights and biases values, it's necessary to know how many weights and biases there are. The demo program implements the static method NumWeights to calculate and return this number. Recall that the 2-(4-2-2)-3 demo network has $(2 \times 4) + (4 \times 2) + (2 \times 2) + (2 \times 3) = 26$ weights and $4 + 2 + 2 + 3 = 11$ biases. The key code in method NumWeights, which calculates the number of input-to-hidden, hidden-to-hidden and hidden-to-output weights is:

```
int ihWts = numInput * numHidden[0];
int hhWts = 0;
for (int j = 0; j < numHidden.Length - 1; ++j) {
    int rows = numHidden[j];
    int cols = numHidden[j + 1];
    hhWts += rows * cols;
}
int hoWts = numHidden[numHidden.Length - 1] * numOutput;
```

Instead of returning the total number of weights and biases as method NumWeights does, you might want to consider returning the number of weights and biases separately, in a two-cell integer array.

Setting Weights and Biases

A non-demo DNN typically initializes all weights and biases to small random values. The demo program sets the 26 weights to 0.01 through 0.26, and the biases to 0.27 through 0.37 using class method SetWeights. The definition begins with:

```
public void SetWeights(double[] wts)
{
    int nw = NumWeights(this.nInput, this.nHidden, this.nOutput);
    if (wts.Length != nw)
        throw new Exception("Bad wts[] length in SetWeights()");
    int ptr = 0;
    ...
}
```

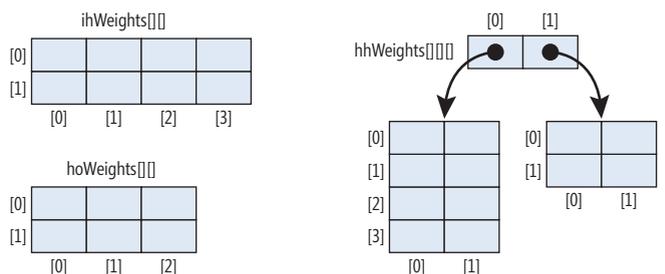


Figure 4 Weights and Biases Data Structures

Manipulating Files?

APIs to view, export, annotate, compare, sign, automate and search documents in your applications.

GroupDocs.Total

GroupDocs.Viewer

GroupDocs.Annotation

GroupDocs.Conversion

GroupDocs.Comparison

GroupDocs.Signature

GroupDocs.Assembly

GroupDocs.Metadata

GroupDocs.Search

GroupDocs.Text



Try for Free



.NET Libraries



Java Libraries



Cloud APIs

Contact Us:

US: +1 903 306 1676

EU: +44 141 628 8900

AU: +61 2 8006 6987

sales@asposeptyltd.com

Visit us at www.groupdocs.com

Input parameter `wts` holds the values for the weights and biases, and is assumed to have the correct `Length`. Variable `ptr` points into the `wts` array. The demo program has very little error checking in order to keep the main ideas as clear as possible. The input-to-first-hidden-layer weights are set like so:

```
for (int i = 0; i < nInput; ++i)
    for (int j = 0; j < hNodes[0].Length; ++j)
        ihWeights[i][j] = wts[ptr++];
```

Next, the hidden-to-hidden weights are set:

```
for (int h = 0; h < nLayers - 1; ++h)
    for (int j = 0; j < nHidden[h]; ++j) // From
        for (int jj = 0; jj < nHidden[h+1]; ++jj) // To
            hhWeights[h][j][jj] = wts[ptr++];
```

If you're not accustomed to working with multi-dimensional arrays, the indexing can be quite tricky. A diagram of the weights and biases data structures is essential (well, for me, anyway). The last-hidden-layer-to-output weights are set like this:

```
int hi = this.nLayers - 1;
for (int j = 0; j < this.nHidden[hi]; ++j)
    for (int k = 0; k < this.nOutput; ++k)
        hoWeights[j][k] = wts[ptr++];
```

This code uses the fact that if there are `nLayers` hidden (3 in the demo), then the index of the last hidden layer is `nLayers-1`. Method `SetWeights` concludes by setting the hidden node biases and the output node biases:

```
...
for (int h = 0; h < nLayers; ++h)
    for (int j = 0; j < this.nHidden[h]; ++j)
        hbases[h][j] = wts[ptr++];

for (int k = 0; k < nOutput; ++k)
    obases[k] = wts[ptr++];
}
```

Computing the Output Values

The definition of class method `ComputeOutputs` begins with:

```
public double[] ComputeOutputs(double[] xValues)
{
    for (int i = 0; i < nInput; ++i)
        iNodes[i] = xValues[i];
    ...
}
```

The input values are in array parameter `xValues`. Class member `nInput` holds the number of input nodes and is set in the class constructor. The first `nInput` values in `xValues` are copied into the input nodes, so `xValues` is assumed to have at least `nInput` values in the first cells. Next, the current values in the hidden and output nodes are zeroed-out:

```
for (int h = 0; h < nLayers; ++h)
    for (int j = 0; j < nHidden[h]; ++j)
        hNodes[h][j] = 0.0;

for (int k = 0; k < nOutput; ++k)
    oNodes[k] = 0.0;
```

The idea here is that the sum of products term will be accumulated directly into the hidden and output nodes, so these nodes must be explicitly reset to 0.0 for each method call. An alternative is to declare and use local arrays with names like `hSums[][]` and `oSums[]`. Next, the values of the nodes in the first hidden layer are calculated:

```
for (int j = 0; j < nHidden[0]; ++j) {
    for (int i = 0; i < nInput; ++i)
        hNodes[0][j] += ihWeights[i][j] * iNodes[i];
    hNodes[0][j] += hbases[0][j]; // Add the bias
    hNodes[0][j] = Math.Tanh(hNodes[0][j]); // Activation
}
```

The code is pretty much a one-one mapping of the mechanism described earlier. The built-in `Math.Tanh` is used for hidden node activation. As I mentioned, important alternatives are the logistic sigmoid function and the rectified linear unit (ReLU) functions, which I'll explain in a future article. Next, the remaining hidden-layer nodes are calculated:

```
for (int h = 1; h < nLayers; ++h) {
    for (int j = 0; j < nHidden[h]; ++j) {
        for (int jj = 0; jj < nHidden[h-1]; ++jj)
            hNodes[h][j] += hhWeights[h-1][jj][j] * hNodes[h-1][jj];
        hNodes[h][j] += hbases[h][j];
        hNodes[h][j] = Math.Tanh(hNodes[h][j]);
    }
}
```

This is the trickiest part of the demo program, mostly due to the multiple array indexes required. Next, the pre-activation sum-of-products are calculated for the output nodes:

```
for (int k = 0; k < nOutput; ++k) {
    for (int j = 0; j < nHidden[nLayers - 1]; ++j)
        oNodes[k] += hoWeights[j][k] * hNodes[nLayers - 1][j];
    oNodes[k] += obases[k]; // Add bias
}
```

Method `ComputeOutputs` concludes by applying the softmax activation function, returning the computed output values in a separate array:

```
...
double[] retResult = Softmax(oNodes);
for (int k = 0; k < nOutput; ++k)
    oNodes[k] = retResult[k];
return retResult;
}
```

The `Softmax` method is a static helper. See the accompanying code download for details. Notice that because softmax activation requires all the values that will be activated (in the denominator term), it's more efficient to compute all softmax values at once instead of separately. The final output values are stored into the output nodes and are also returned separately for calling convenience.

Wrapping Up

There has been enormous research activity and many breakthroughs related to deep neural networks over the past few years. Specialized DNNs such as convolutional neural networks, recurrent neural networks, LSTM neural networks and residual neural networks are very powerful but very complex. In my opinion, understanding how basic DNNs operate is essential for understanding the more complex variations.

In a future article, I'll explain in detail how to use the back-propagation algorithm (arguably the most famous and important algorithm in machine learning) to train a basic DNN. Back-propagation, or at least some form of it, is used to train most DNN variations, too. This explanation will introduce the concept of the vanishing gradient, which in turn will explain the design and motivation of many of the DNNs now being used for very sophisticated prediction systems. ■

Dr. James McCaffrey works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products, including Internet Explorer and Bing. Dr. McCaffrey can be reached at jamccaff@microsoft.com.

THANKS to the following Microsoft technical experts who reviewed this article: Li Deng, Pingjun Hu, Po-Sen Huang, Kirk Li, Alan Liu, Ricky Loynd, Baochen Sun, Henrik Turbell.

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

AUGUST 7 - 11, 2017

MICROSOFT HEADQUARTERS

REDMOND, WA



**PLUG IN TO NEW KNOWLEDGE
@ THE SOURCE**



WHAT SETS TECHMENTOR APART?

- + Immediately usable IT education
- + Training 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
- + Intimate setting, where your voice is heard, making it a viable alternative to huge, first-party conferences
- + Experience life @ Microsoft Headquarters for a full week

**YOU OWE IT TO YOURSELF, YOUR COMPANY AND
YOUR CAREER TO BE AT TECHMENTOR REDMOND 2017!**

HOT TRAINING TOPICS INCLUDE:

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



REGISTER NOW



**SAVE \$300 THROUGH AUGUST 7
MUST USE DISCOUNT CODE TMEB01**

[TECHMENTOREVENTS.COM/REDMOND]

EVENT SPONSOR:  Microsoft

GOLD SPONSOR: 

SUPPORTED BY:  

PRODUCED BY: 



How To Be MEAN: Up-Angular-izing

Welcome back again, MEANers.

It's been two years since I started this particular series on the MEAN (Mongo, Express, Angular, Node) stack. And, as was bound to happen, various parts of the MEAN stack have evolved since the series started. Most of these updates (specifically the Node, Express and Mongo versions) are transparent, and adopting them is a non-event: Just upgrade the underlying bits and everything works.

But Angular upgrades have been of some concern to the Web front-end world for a while, particularly because the substantive changes between AngularJS (v1) and Angular (v2 and beyond) created some serious backward-compatibility issues. (I use the term "backward compatibility" loosely here, because the backward-compatibility story for v1 to v2 was essentially, "Rewrite the whole thing—trust us, it'll be great!") Thus, it was with some amount of consternation that the Angular world was watching for the first major update to Angular, and when that update was announced to be a major version enhancement, anxiety mounted.

Turns out, while we were busy writing the front end of the sample application, the Angular team did what they were supposed to do—release a new version of Angular into the world. That means it's time to take a moment, bite the bullet, and upgrade the application to the new version of Angular: v4. (The Angular team decided to skip 3 and move straight to 4.) Spoiler alert: This turns out to be far, far less painful than people imagined it might be, and offers a lot of hope regarding future Angular updates—which is good because the Angular team has promised that they're going to release cadence much more in line with traditional open source projects. Which means, bluntly, a lot of small, incremental upgrades released much more quickly (every 6 months) than what's been the norm so far.

Upgrading Angular

Fundamentally, upgrading to Angular 4 means using the Node Package Manager (npm) to update the npm packages in use to the latest versions. This takes the form of the too-familiar "npm install" command, using a version tag ("@latest") for each package and the "--save" argument to capture the latest version into the application's package.json file. For those running on a *nix system (Linux or macOS, typically), the command takes the following form, all of which should be typed on one line:

```
npm install @angular/{common,compiler,compiler-  
cli,core,forms,http,platform-browser,platform-browser-dynamic,platform-  
server,router,animations}@latest typescript@latest --save
```

The *nix command shells allow for the various packages to be captured under the "{/}" pairs, even though technically each one

is named "@angular/common," "@angular/compiler" and so on. For those of you on Windows, you get this slightly longer version:

```
npm install @angular/common@latest @angular/compiler@latest @angular/  
compiler-cli@latest @angular/core@latest @angular/forms@latest @angular/  
http@latest @angular/platform-browser@latest @angular/platform-browser-  
dynamic@latest @angular/platform-server@latest @angular/router@latest @  
angular/animations@latest typescript@latest --save
```

Once the "npm install" is finished executing, for all intents and purposes the upgrade is done. Simply run the application using "ng serve" again, and everything should be back to running status.

Angular 2-to-4 Pain Points

The Angular team has admitted that it's not always a smooth transition—however, the release notes take care to point out that most of the pain (apparently) is localized to the use of animations, which is a subject I haven't explored yet. Specifically, the team removed animations from @angular/core, and dropped them into its own Node package, @angular/animations (which you can see in the previous "npm install" command). That way, if the application doesn't use animations, it doesn't have to carry along the code of animations in the core package.

Fundamentally, upgrading to Angular 4 means using the Node Package Manager (npm) to update the npm packages in use to the latest versions.

Angular 4 New Features

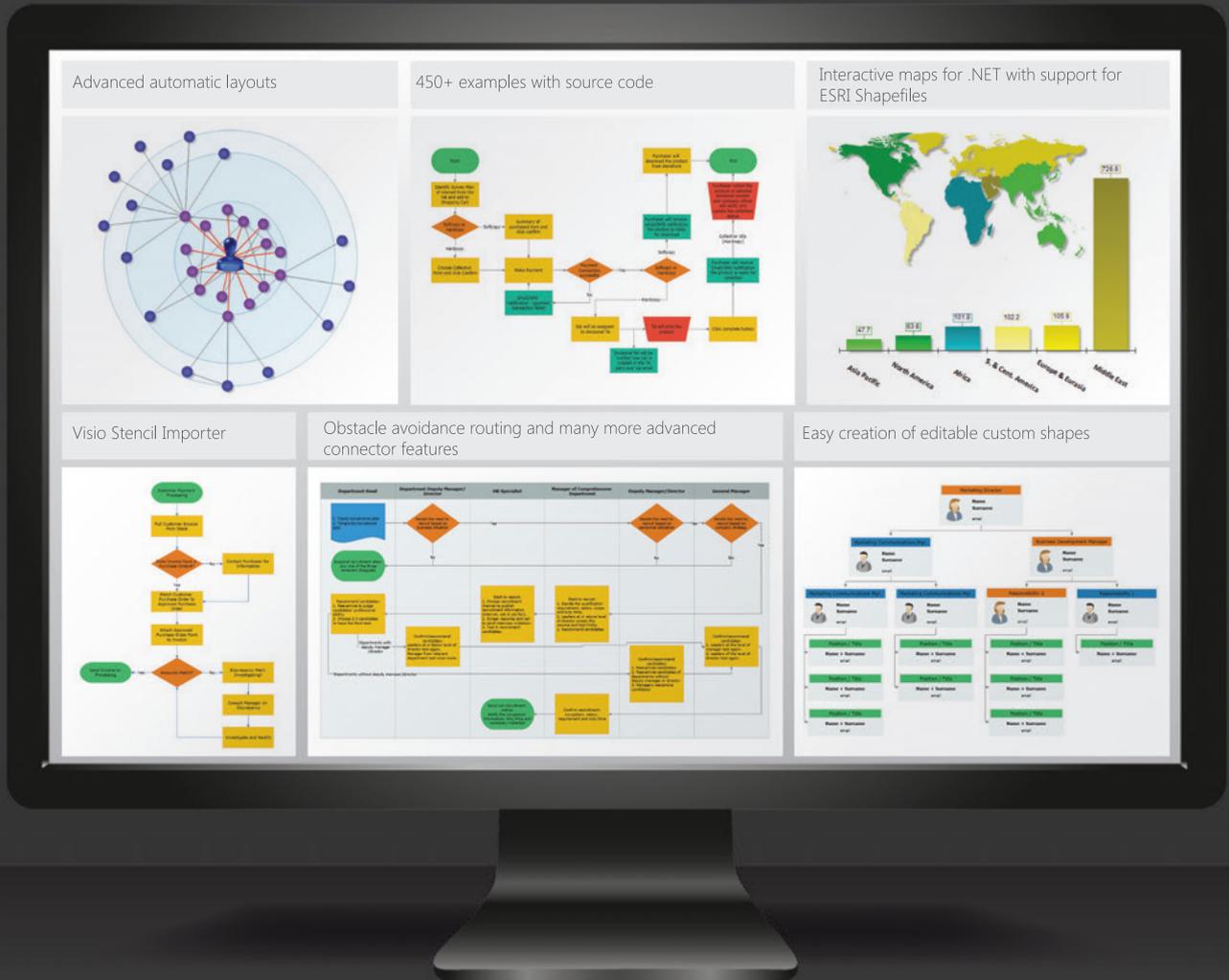
The Angular 4 release notes carry the full weight of the story, but there are a few things in particular worth calling out.

First, the Angular team is focused on reducing the size/weight footprint of the Angular libraries. This is good for obvious reasons, particularly for those users who aren't on high-speed fiber connections with the rest of the world. The Angular team says they're not done, either, so expect that each successive Angular release will seek to decrease its footprint even further.

In the same spirit, the Angular team has reduced the overall size of the generated codebehind view templates, up to 60 percent.

**EXTENSIBLE INTERACTIVITY AND EDITING OPTIONS • AUTOMATIC GRAPH AND TREE LAYOUTS
 IMPORT FROM DATABASE, VISIO AND ESRI DATA • HUNDREDS OF GENUINE CLIPART SHAPES
 450+ EXAMPLES WITH SOURCE CODE**

NEVRON DIAGRAM FOR .NET



ADVANCED & FAST

Part of  **NEVRON VISION for .NET**

Learn more at www.nevron.com today

Microsoft **ASP.net** | Microsoft **Silverlight** | **WPF** | **WinForms** | **SharePoint** | Microsoft **SQL Server** | **Xamarin**

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.



Again, this means that the application you build will be that much smaller and lighter.

Second, the team improved the “*ngIf” and “*ngFor” directives used in view templates for branching and iteration scenarios, respectively. You haven’t seen those yet, so the new features won’t be apparent yet, but you’ll see them soon, so hang in there.

Last, the Angular team also brought the Angular libraries up to the latest versions of TypeScript (2.2), which includes better nullable checking, some better type support for ECMAScript (ES) 2015-style mixings, and an “object” type to represent a type that’s the base type of all declared types in TypeScript, similar to the role that System.Object serves in much .NET code. This implicitly also brings support for TypeScript 2.1, which has some interesting features on its own, like the “keyof” operator, mapped types (which provides the utility types Partial, Readonly, Pick and Record), and support for the “spread” and “rest” operators from ES 2015. All of this is well beyond the scope of Angular itself, but any good TypeScript tutorial (or the TypeScript Web site itself) will explain their use. Fundamentally, these won’t change the code that you write when writing Angular, at least not right away, but as these features get used more in the Angular library, they might start finding their way into the surface area of the Angular API. That likely won’t happen for a while, however, so for the moment, the biggest thing to keep in mind is that Angular is keeping up with the evolution of TypeScript.

In the same spirit, the Angular team has reduced the overall size of the generated codebehind view templates, up to 60 percent.

Wrapping Up

Hopefully I’ve helped you understand that doing this upgrade costs you almost nothing to do—that’s the best kind of version update. More important, it’s refreshing to know that as Angular applications grow and evolve, the required work to keep them up-to-date with the latest versions of Angular is (for the moment, anyway) trivial.

How To Be MEAN: Two Years On

While working on this column, *MSDN Magazine* Editor in Chief Michael Desmond pointed out that my How To Be MEAN series was turning 2 years old as of this issue. How is it that I’m still working in the MEAN mines? Some of it has to do with the fact that this series is attacking a rather large subject—a complete soup-to-nuts, front-end-to-data-storage, REST API middleware-based platform, rather than just a library or framework. But some of it has to do with the nature of the MEAN stack itself.

You see, the MEAN platform is different from the .NET Framework platform not in terms of what it provides—both have a programming language, an HTTP library/framework for receiving

JSON data submitted, drivers for accessing databases and so on. Rather, it differs in terms of what it doesn’t provide. That is to say, the MEAN platform, building on top of the Node.js platform stresses a sense of “minimalism” that the .NET platform doesn’t.

That might sound like a slight to one or the other platform; that somehow Node.js isn’t “fully baked” or that .NET is “too heavy.” No such value judgement is intended. But where .NET emerged from Microsoft and continues to be heavily driven by what the .NET Framework team has built over the years, the Node.js platform has been bolted together by libraries built by hundreds of teams and thousands of developers from all across the world. There are pros and cons to each approach—but that’s not the direction I’m headed with this.

The fact is both platforms are available to you, at your discretion. And even just two years ago, the idea of Microsoft being a platform by which developers could use either .NET or Node.js—or even Java or PHP—for building applications on or near the Microsoft OS (or cloud platform) seemed ludicrous. There were signs that suggested that Microsoft might reach this kind of “all platforms created equal” mentality, but the company’s history suggested we might see an approach where .NET would be first among those equals.

Consider this for a moment: The “A” in the MEAN stack stands for Angular. When I began this series, Angular was not the powerhouse, rich-client, single-page application (SPA) platform that it is today—it was but one of several potential bets that you might make on the JavaScript front-end landscape. Angular has seen a definite rise in interest, and the pages of this magazine have been decorated with numerous references to Angular, both within the confines of this column and in feature pieces written by others.

What’s remarkable is that this interest is in a front-end technology written in the open source world by a team that not only doesn’t work for Microsoft, but works for one of Microsoft’s competitors. Yet it uses the open source TypeScript language developed by Microsoft. It’s enough to make your head spin.

The MEAN stack, and the coverage of MEAN in this magazine in many ways articulate everything about “the new Microsoft.” It’s a stellar demonstration of how the Microsoft of 2017 is so entirely different from the Microsoft of 2007 or 2000. The Microsoft that valued competition over cooperation and community is long gone. The company before us today certainly competes, but not with its community. The Microsoft of 2017 wants you to use the technology stack of your choice, ideally within its cloud or on its OS, but if you have a different choice than that, well, that’s your choice.

At the end of the day, the MEAN stack is “just” a stack made up of three parts (MongoDB, Angular and Node.js/Express) that can interoperate with one another. And the fact that Microsoft not only embraces that, but encourages it, tells you just how far things have come from where it was before.

Kind of makes you wonder what the next few years have in store for us, doesn’t it? Happy coding! ■

TED NEWARD is a Seattle-based polytechnology consultant, speaker and mentor, currently working as the director of developer relations at Smartsheet.com. He has written more than 100 articles, authored and coauthored a dozen books, and works all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.

Free AppFabric Wrapper
Quick AppFabric Migration to NCache



Extreme Performance Linear Scalability



Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing



NoSQL Database

- Schema-Free JSON Documents
- Multiple Shards & Data Replication
- Powerful SQL, LINQ, ADO.NET

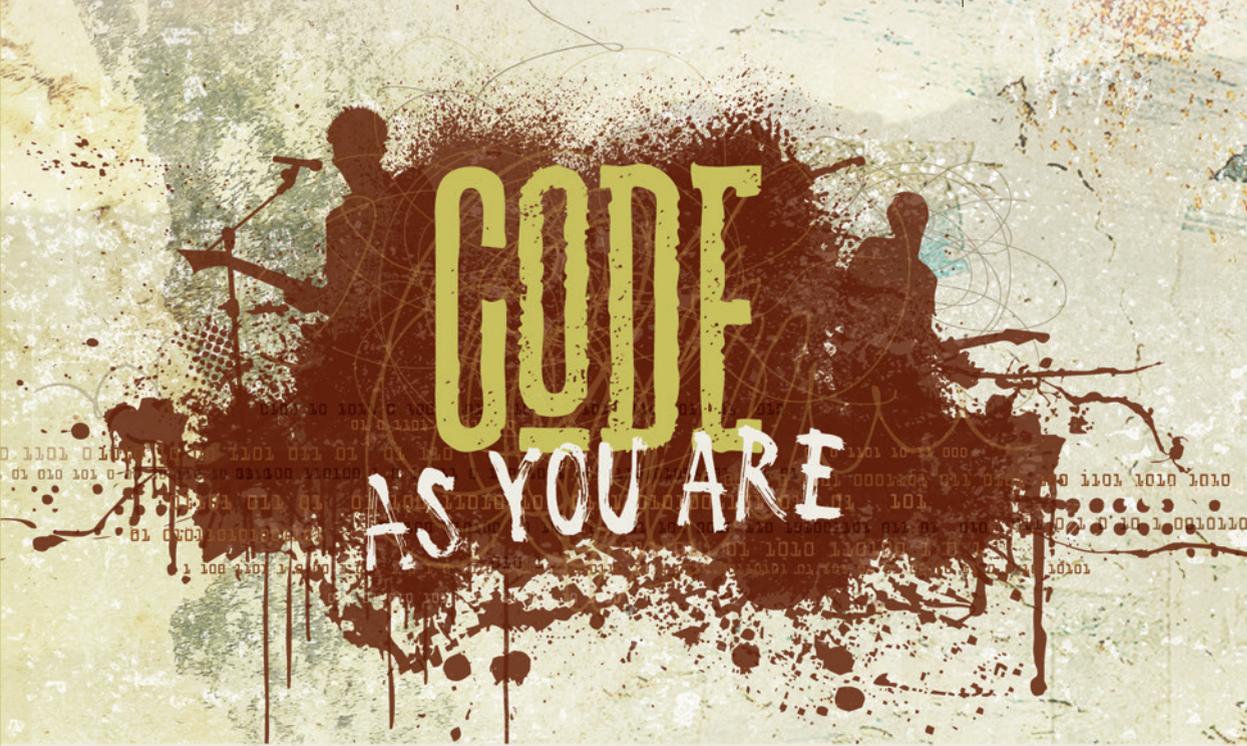
100% Native .NET

Open Source

Visual Studio **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND
AUGUST 14-18, 2017
 MICROSOFT HEADQUARTERS



START TIME	END TIME
9:00 AM	6:00 PM
START TIME	END TIME
8:00 AM	12:00 PM
12:00 PM	2:00 PM
2:00 PM	5:30 PM
7:00 PM	9:00 PM
START TIME	END TIME
8:00 AM	9:15 AM
9:30 AM	10:45 AM
10:45 AM	11:15 AM
11:15 AM	12:15 PM
12:15 PM	1:30 PM
1:30 PM	2:45 PM
3:00 PM	4:15 PM
4:15 PM	5:45 PM
START TIME	END TIME
8:00 AM	9:15 AM
9:30 AM	10:45 AM
11:00 AM	12:00 PM
12:00 PM	1:30 PM
1:30 PM	2:45 PM
2:45 PM	3:15 PM
3:15 PM	4:30 PM
6:15 PM	8:30 PM
START TIME	END TIME
8:00 AM	9:15 AM
9:30 AM	10:45 AM
11:00 AM	12:15 PM
12:15 PM	2:15 PM
2:15 PM	3:30 PM
3:45 PM	5:00 PM
START TIME	END TIME
8:00 AM	5:00 PM

JOIN US AT MICROSOFT HEADQUARTERS THIS SUMMER

SUNDAY, AUG 13: PRE-CON HANDS-ON LABS

Choose From:

- Angular
- Dev Ops with ASP.NET Core/EF Core
- SQL Server 2016

NEW!
 Only **\$695!**

SPACE IS LIMITED

- Rub elbows with blue badges
- Experience life on campus
- Enjoy lunch in the Commons and visit the Company Store
- And More!



Scott Hanselman, Keynote Speaker

FULL TRACK OF MICROSOFT SESSIONS NOW UPDATED!

There are many perks to attending training at Microsoft Headquarters, and one of the biggest is getting to hear from Microsoft insiders who are "in the trenches", working with developers just like you on a daily basis.

Microsoft Speakers are noted with a 

ROCK YOUR CODE TOUR 2017

Register by August 14 and Save **\$300!***

Use promo code RDEB01

REGISTER NOW

*Only available on 3- and 5-day packages.

EVENT PARTNER



GOLD SPONSORS



SILVER SPONSOR



SUPPORTED BY



PRODUCED BY



REDMOND AGENDA AT-A-GLANCE

ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server
--------------	-----------------	------------------------	---------------	--------------------	--------------------------------	------------	------------

NEW Full Day Hands-On Labs: Sunday, August 13, 2017 (Separate entry fee required)

HOL01 Full Day Hands-On Lab: Busy Developer's HOL on Angular - <i>Ted Neward</i>	HOL02 Full Day Hands-On Lab: DevOps with ASP.NET Core and EF Core - <i>Benjamin Day & Brian Randell</i>	HOL03 Full Day Hands-On Lab: Developer Dive into SQL Server 2016 - <i>Leonard Lobel</i>
---	--	--

Visual Studio Live! Pre-Conference Workshops: Monday, August 14, 2017 (Separate entry fee required)

M01 Workshop: Modern Security Architecture for ASP.NET Core - <i>Brock Allen</i>	M02 Workshop: Distributed Cross-Platform Application Architecture - <i>Jason Bock & Rockford Lhotka</i>	M03 Workshop: Big Data, BI and Analytics on The Microsoft Stack - <i>Andrew Brust</i>
---	--	--

Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center

M01 Workshop Continues - <i>Brock Allen</i>	M02 Workshop Continues - <i>Jason Bock & Rockford Lhotka</i>	M03 Workshop Continues - <i>Andrew Brust</i>
--	---	---

Dine-A-Round Dinner

Visual Studio Live! Day 1: Tuesday, August 15, 2017

T01 Go Mobile With C#, Visual Studio, and Xamarin - <i>James Montemagno</i>	T02 Angular 101: Part 1 - <i>Deborah Kurata</i>	T03 New SQL Server 2016 Security Features for Developers - <i>Leonard Lobel</i>	T04 Building an Agile Culture that Scales - <i>Aaron Bjork</i>	T05 The Future of C# - <i>Dustin Campbell & Mads Torgersen</i>
T06 Building Connected and Disconnected Mobile Apps - <i>James Montemagno</i>	T07 Angular 101: Part 2 - <i>Deborah Kurata</i>	T08 No Schema, No Problem! Introduction to Azure DocumentDB - <i>Leonard Lobel</i>	T09 Getting to the Core of .NET Core - <i>Adam Tuliper</i>	T10 Building Apps with Microsoft Graph and Visual Studio - <i>Robert Green</i>

Sponsored Break - Visit Exhibitors

KEYNOTE: Microsoft's Open Source Developer Journey - Scott Hanselman, Principal Community Architect for Web Platform and Tools, Microsoft

Lunch - Visit Exhibitors

T11 Take the Tests: Can You Evaluate Good and Bad Designs? - <i>Billy Hollis</i>	T12 Assembling the Web—A Tour of WebAssembly - <i>Jason Bock</i>	T13 Unit Testing & Test-Driven Development (TDD) for Mere Mortals - <i>Benjamin Day</i>	T14 Mobile DevOps with the Microsoft Stack - <i>Abel Wang</i>	T15 Azure for .NET Developers: In Plain English - <i>Michael Crump</i>
T16 A Developers Introduction to HoloLens - <i>Billy Hollis & Brian Randell</i>	T17 Spans, Memory, and Channels—Making .NET Code Fast - <i>Jason Bock</i>	T18 Developing for Windows and Linux Side by Side - <i>Gilles Khouzam</i>	T19 Entity Framework Core for Enterprise Applications - <i>Benjamin Day</i>	T20 Debugging Tips and Tricks for Visual Studio - <i>Kaycee Anderson</i>

Microsoft Ask the Experts & Exhibitor Reception Sponsored by smartsheet

Visual Studio Live! Day 2: Wednesday, August 16, 2017

W01 Roll Your Own Dashboard in XAML - <i>Billy Hollis</i>	W02 Migrating to ASP.NET Core—A True Story - <i>Adam Tuliper</i>	W03 Hacker Trix - Learning from OWASP Top 10 - <i>Mike Benkovich</i>	W04 Distributed Architecture: Microservices and Messaging - <i>Rockford Lhotka</i>	W05 Architecting Big Data Solutions with Azure - <i>Michael Rys</i>
W06 Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - <i>Laurent Bugnion</i>	W07 User Authentication for ASP.NET Core MVC Applications - <i>Brock Allen</i>	W08 From Containers to Data in Motion, Tour d'Azure 2017 - <i>Mike Benkovich</i>	W09 ASP.NET Core 2.0 - <i>Jass Bagga</i>	W10 Agile: You Keep Using That Word... - <i>Philip Japikse</i>

GENERAL SESSION: Amplifying Human Ingenuity with Microsoft AI - Paul Stubbs, Director of Product Marketing for AI and Bots, Microsoft

Birds-of-a-Feather Lunch - Visit Exhibitors

W11 Building Cross-platform App. Dev. with CLS.NET - <i>Rockford Lhotka</i>	W12 Securing Web APIs in ASP.NET Core - <i>Brock Allen</i>	W13 Tactical DevOps with VSTS - <i>Brian Randell</i>	W14 TypeScript and the Future of JavaScript - <i>Jordan Matthiesen & Bowden Kelly</i>	W15 Agile Failures: Stories from The Trenches - <i>Philip Japikse</i>
--	---	---	--	--

Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win)

W16 Building Truly Universal Applications with Windows, Xamarin and MVVM - <i>Laurent Bugnion</i>	W17 Integrating AngularJS & ASP.NET MVC - <i>Miguel Castro</i>	W18 Get Started with Git and GitHub - <i>Robert Green</i>	W19 SOLID—The Five Commandments of Good Software - <i>Chris Klug</i>	W20 Using Angular 2, JavaScript, and TypeScript to Build Fast and Secure Mobile Apps - <i>Jordan Matthiesen</i>
--	---	--	---	--

Set Sail! VSLive's Seattle Sunset Cruise - Advanced Reservation & \$10 Fee Required

Visual Studio Live! Day 3: Thursday, August 17, 2017

TH01 Lessons Learned from Real World Xamarin.Forms Projects - <i>Nick Landry</i>	TH02 Build Real-Time Websites and Apps with SignalR - <i>Rachel Appel</i>	TH03 "Aurelia vs. "Just Angular" a.k.a. "The Framework Formerly Known as Angular 2" - <i>Chris Klug</i>	TH04 Go Serverless with Azure Functions - <i>Eric D. Boyd</i>	TH05 Git at Microsoft Scale - <i>Edward Thomson</i>
TH06 Creating Great Looking Android Applications Using Material Design - <i>Kevin Ford</i>	TH07 Database Lifecycle Management and the SQL Server Database - <i>Brian Randell</i>	TH08 Hard Core ASP.NET Core - <i>Rachel Appel</i>	TH09 Breaking Down Walls with Modern Identity - <i>Eric D. Boyd</i>	TH10 Microsoft Set List: Details Dropping Soon
TH11 Software Engineering in an Agile Environment - <i>David Corbin</i>	TH12 Bots are the New Apps: Building Bots with ASP.NET WebAPI & Language Understanding - <i>Nick Landry</i>	TH13 Power BI: Analytics for Desktop, Mobile and Cloud - <i>Andrew Brust</i>	TH14 Enriching MVC Sites with Knockout JS - <i>Miguel Castro</i>	TH15 What's New in Visual Studio 2017 for C# Developers - <i>Kasey Uhlenhuth</i>

Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center

TH16 Classic Software Design Principles and Why They Are Still Important - <i>David Corbin</i>	TH17 Getting Started with Aurelia - <i>Brian Noyes</i>	TH18 Big Data with Hadoop, Spark and Azure HDInsight - <i>Andrew Brust</i>	TH19 Extend and Customize the Visual Studio Environment - <i>Walt Ritscher</i>	TH20 Serverless with Azure Functions—Scale Dynamically and Pay per Execution - <i>Donna Malayeri</i>
TH21 End-to-End Dependency Injection & Testable Code - <i>Miguel Castro</i>	TH22 Everything You Need to Know About Package Management - <i>Alex Mullans</i>	TH23 Continuous Integration and Deployment for Mobile using Azure Services - <i>Kevin Ford</i>	TH24 Windows Package Management with NuGet and Chocolatey - <i>Walt Ritscher</i>	TH25 Securing Client Apps with IdentityServer - <i>Brian Noyes</i>

Visual Studio Live! Post-Conference Workshops: Friday, August 18, 2017 (Separate entry fee required)

F01 Workshop: Building Modern Web Apps with Azure - <i>Eric D. Boyd</i>	F02 Workshop: Data-Centric Single Page Apps with Aurelia, Breeze, and Web API - <i>Brian Noyes</i>
--	---

Speakers and sessions subject to change

CONNECT WITH US



vslive.com/redmondmsdn



C# 7.0: Tuples Explained

Back in November, in the `Connect()`; special issue, I provided an overview of C# 7.0 (msdn.microsoft.com/magazine/mt790178), in which I introduced tuples. In this article, I delve into tuples again, covering the full breadth of the syntax options.

To begin, let's consider the question: Why tuples? On occasion, you'll likely find it useful to combine data elements. Suppose, for example, you're working with information about countries, such as the poorest country in the world in 2017: Malawi, whose capital is Lilongwe, with a gross domestic product (GDP) per capita of \$226.50. You could obviously declare a class for this data, but it doesn't really represent your typical noun/object. It's seemingly more a collection of related pieces of data than it is an object. Surely, if you were going to have a `Country` object, for example, it would have considerably more data than just properties for the `Name`, `Capital` and `GDP per capita`. Alternatively, you could store each data element in individual variables, but the result would be no association between the data elements; \$226.50 would have no association with Malawi except perhaps by a common suffix or prefix in the variable names. Another option would be to combine all the data into a single string—with the disadvantage that to work with each data element individually would require parsing it out. A final approach might be to create an anonymous type, but that, too, has limitations; enough, in fact, that tuples could potentially replace anonymous types entirely. I'll leave this topic until the end of the article.

The best option might be the C# 7.0 tuple, which, at its simplest, provides a syntax that allows you to combine the assignment of multiple variables, of varying types, in a single statement:

```
(string country, string capital, double gdpPerCapita) =  
("Malawi", "Lilongwe", 226.50);
```

In this case, I'm not only assigning multiple variables, but declaring them as well.

However, tuples have several other additional syntax possibilities, each shown in **Figure 1**.

In the first four examples, and although the right-hand side represents a tuple, the left-hand side still represents individual variables that are assigned together using *tuple syntax*, which involves two or more elements separated by commas and associated with parentheses. (I use the term *tuple syntax* because the underlying data type the compiler generates on the left-hand side isn't technically a tuple.) The result is that although I start with values combined as a

tuple on the right, the assignment to the left deconstructs the tuple into its constituent parts. In example 2, the left-hand-side assignment is to pre-declared variables. However, in examples 1, 3 and 4, the variables are declared within the tuple syntax. Given that I'm only declaring variables, the naming and casing convention follows the generally accepted Framework Design Guidelines—"Do use camelCase for local variable names," for example.

Note that although implicit typing (`var`) can be distributed across each variable declaration within the tuple syntax, as shown in example 4, you can't do the same with an explicit type (such as `string`). In this case, you're actually declaring a tuple type, not just using tuple syntax and, therefore, you'll need to add a reference to the `System.ValueType` NuGet package—at least until .NET Standard 2.0. Because tuples allow each item to be a different data type, distributing the explicit type name across all elements wouldn't necessarily work unless all the item data types were identical (and even then, the compiler doesn't allow it).

In example 5, I declare a tuple on the left-hand side and then assign the tuple on the right. Note that the tuple has named items—names you can then reference to retrieve the item values back out of the tuple. This is what enables the `countryInfo.Name`, `countryInfo.Capital`, and `countryInfo.GdpPerCapita` syntax in the `System.Console.WriteLine` statement. The result of the tuple declaration on the left is a grouping of the variables into a single variable (`countryInfo`) from which you can then access the constituent parts. This is useful because you can then pass this single variable around to other methods and those methods will also be able to access the individual items within the tuple.

As already mentioned, variables defined using tuple syntax use camelCase. However, the convention for tuple item names isn't well-defined. Suggestions include using parameter-naming conventions when the tuple behaves like a parameter—such as when returning multiple values that before tuple syntax would've used out parameters. The alternative is to use PascalCase, following the naming convention for public fields and properties. I strongly favor the latter approach in accordance with the Capitalization Rules for Identifiers (itl.tc/caprfi). Tuple item names are rendered as members of the tuple and the convention for all (public) members (which are potentially accessed using a dot operator) is PascalCase.

Example 6 provides the same functionality as example 5, although it uses named tuple items on the right-hand side tuple value and an implicit type declaration on the left. The items' names are persisted

Code download available at itl.tc/MSDN.2017.08.

Spreadsheets Made Easy.



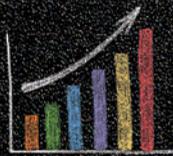
SpreadsheetGear 2017 Released

SpreadsheetGear 2017 adds a new SpreadsheetGear for .NET Standard product, official support for Excel 2013 and Excel 2016, 51 new Excel functions for a total of 449 fully supported functions, full conditional formatting support, enhanced workbook protection and encryption, cell gradient rendering and more.



Support for iOS, Android, Linux, macOS, UWP and more

SpreadsheetGear for .NET Standard enables cross-platform developers to enjoy the same high performance Excel-compatible reporting, charting, calculations and more relied on by thousands of Windows developers for 10+ years.



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.

Download your free fully functional evaluation at SpreadsheetGear.com



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.



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.



Fastest Calculations

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



SpreadsheetGear

to the implicitly typed variable, however, so they're still available for the WriteLine statement. Of course, this opens the possibility that you could name the items on the left-hand side with names that are different from those you use on the right. While the C# compiler allows this, it will issue a warning that the item names on the right will be ignored as those on the left take precedence.

If no item names are specified, the individual elements are still available from the assigned tuple variable. However, the names are Item1, Item2 and so on, as shown in example 7. In fact, the ItemX name is always available on the tuple—even when custom names are provided (see example 8). However, when using IDE tools like any of the recent flavors of Visual Studio that support C# 7.0, the ItemX property will not appear within the IntelliSense dropdown—a good thing because presumably the provided name is preferable.

As shown in example 9, portions of a tuple assignment can be excluded using an underscore; this is called a discard.

Tuples are a lightweight solution for encapsulating data into a single object in the same way that a bag might capture miscellaneous items you pick up from the store. Unlike arrays, tuples contain item data types that can vary virtually without constraint (although pointers aren't allowed), except that they're identified by the code and can't be changed at run time. Also, unlike with arrays, the number of items within the tuple is hardcoded at compile time, as well. Last, you can't add custom behavior to a tuple (extension methods notwithstanding). If you need behavior associated with the encapsulated data, then leveraging object-oriented programming and defining a class is the preferred approach.

The System.ValueTuple<...> Type

The C# compiler generates code that relies on a set of generic value types (structs), such as System.ValueTuple<T1, T2, T3>, as the underlying implementation for the tuple syntax for all tuple instances on the right-hand side of the examples in **Figure 1**. Similarly, the same set of System.ValueTuple<...> generic value types is used for the left-hand-side data type starting with example 5. As you'd expect with a tuple type, the only methods included are those related to comparison and equality. However, perhaps unexpectedly, there

are no properties for ItemX, but rather read-write fields (seemingly breaking the most basic of .NET Programming Guidelines as explained at itl.tc/CS7TuplesBreaksGuidelines).

In addition to the programming guidelines discrepancy, there's another behavioral question that arises. Given that the custom item names and their types aren't included in the System.ValueTuple<...> definition, how is it possible that each custom item name is seemingly a member of the System.ValueTuple<...> type and accessible as a member of that type?

What's surprising (particularly for those familiar with the anonymous type implementation) is that the compiler doesn't generate underlying Common Intermediate Language (CIL) code for the members corresponding to the custom names. However, even without an underlying member with the custom name, there is (seemingly) from the C# perspective, such a member.

Figure 1 Sample Code for Tuple Declaration and Assignment

Example	Description	Example Code
1.	Assigning a tuple to individually declared variables.	<pre>(string country, string capital, double gdpPerCapita) = ("Malawi", "Lilongwe", 226.50); System.Console.WriteLine(\$"{@"The poorest country in the world in 2017 was { country}, {capital}: {gdpPerCapita}"});</pre>
2.	Assigning a tuple to individually declared variables that are pre-declared.	<pre>string country; string capital; double gdpPerCapita; (country, capital, gdpPerCapita) = ("Malawi", "Lilongwe", 226.50); System.Console.WriteLine(\$"{@"The poorest country in the world in 2017 was { country}, {capital}: {gdpPerCapita}"});</pre>
3.	Assigning a tuple to individually declared and implicitly typed variables.	<pre>(var country, var capital, var gdpPerCapita) = ("Malawi", "Lilongwe", 226.50); System.Console.WriteLine(\$"{@"The poorest country in the world in 2017 was { country}, {capital}: {gdpPerCapita}"});</pre>
4.	Assigning a tuple to individually declared variables that are implicitly typed with a distributive syntax.	<pre>var (country, capital, gdpPerCapita) = ("Malawi", "Lilongwe", 226.50); System.Console.WriteLine(\$"{@"The poorest country in the world in 2017 was { country}, {capital}: {gdpPerCapita}"});</pre>
5.	Declaring a named item tuple and assigning it tuple values and then accessing the tuple items by name.	<pre>(string Name, string Capital, double GdpPerCapita) countryInfo = ("Malawi", "Lilongwe", 226.50); System.Console.WriteLine(\$"{@"The poorest country in the world in 2017 was { countryInfo.Name}, {countryInfo.Capital}: { countryInfo.GdpPerCapita}"});</pre>
6.	Assigning a named item tuple to a single implicitly typed variable that's implicitly typed and then accessing the tuple items by name.	<pre>var countryInfo = (Name: "Malawi", Capital: "Lilongwe", GdpPerCapita: 226.50); System.Console.WriteLine(\$"{@"The poorest country in the world in 2017 was { countryInfo.Name}, {countryInfo.Capital}: { countryInfo.GdpPerCapita}"});</pre>
7.	Assigning an unnamed tuple to a single implicitly typed variable and then accessing the tuple elements by their Item-number property.	<pre>var countryInfo = ("Malawi", "Lilongwe", 226.50); System.Console.WriteLine(\$"{@"The poorest country in the world in 2017 was { countryInfo.Item1}, {countryInfo.Item2}: { countryInfo.Item3}"});</pre>
8.	Assigning a named item tuple to a single implicitly typed variable and then accessing the tuple items by their Item-number property.	<pre>var countryInfo = (Name: "Malawi", Capital: "Lilongwe", GdpPerCapita: 226.50); System.Console.WriteLine(\$"{@"The poorest country in the world in 2017 was { countryInfo.Item1}, {countryInfo.Item2}: { countryInfo.Item3}"});</pre>
9.	Discard portions of the tuple with underscores.	<pre>(string name, _, double gdpPerCapita) countryInfo = ("Malawi", "Lilongwe", 226.50);</pre>

For all the named tuple local variable examples, for example:

```
var countryInfo = (Name: "Malawi", Capital: "Lilongwe", GdpPerCapita: 226.50)
```

it's clearly possible that the names could be known by the compiler for the remainder of the scope of the tuple because that scope is bounded within the member in which it's declared. And, in fact, the compiler (and IDE) quite simply rely on this scope to allow accessing each item by name. In other words, the compiler looks at the item names within the tuple declaration and leverages them to allow code that uses those names within the scope. It's for this reason, as well, that the `ItemX` methods aren't shown in the IDE IntelliSense as available members on the tuple (the IDE simply ignores them and replaces them with the named items).

Determining the item names from when scoped within a member is reasonable for the compiler, but what happens when a tuple is exposed outside the member—such as a parameter or return from a method that's in a different assembly (for which there's possibly no source code available)? For all tuples that are part of the API (whether a public or private API), the compiler adds item names to the metadata of the member in the form of attributes. For example, this:

```
[return: System.Runtime.CompilerServices.TupleElementNames(
    new string[] {"First", "Second"})]
public System.ValueTuple<string, string> ParseNames(string fullName)
{
    // ...
}
```

is the C# equivalent of what the compiler generates for the following:

```
public (string First, string Second) ParseNames(string fullName)
```

On a related note, C# 7.0 doesn't enable the use of custom item names when using the explicit `System.ValueTuple<...>` data type. Therefore, if you replace `var` in Example 8 of **Figure 1**, you'll end up with warnings that each item name will be ignored.

Here are a few additional miscellaneous facts to keep in mind about `System.ValueTuple<...>`:

- There are a total of eight generic `System.ValueTuple` structs corresponding to the possibility of supporting a tuple with up to seven items. For the eighth tuple, `System.ValueTuple<T1, T2, T3, T4, T5, T6, T7, TRest>`, the last type parameter allows specifying an additional value tuple, thus enabling support for n items. If, for example, you specify a tuple with 8 parameters, the compiler will automatically generate a `System.ValueTuple<T1, T2, T3, T4, T5, T6, T7, System.ValueTuple<TSub1>>` as the underlying implementing type. (For completeness, `System.Value<T1>` exists, but will really only be used directly and only as a type. It will never be used directly by the compiler because the C# tuple syntax requires a minimum of two items.)
- There is a non-generic `System.ValueTuple` that serves as a tuple factory with `Create` methods corresponding to each value tuple arity. The ease of using a tuple literal, such as `var t1 = ("Inigo Montoya", 42)`, supersedes the `Create` method at least for C# 7.0 (or later) programmers.

- For all practical purposes, C# developers can essentially ignore `System.ValueTuple` and `System.ValueTuple<T>`.

There's another tuple type that was included with the .NET Framework 4.5—`System.Tuple<...>`. At that time, it was expected to be the core tuple implementation going forward. However, once C# supported tuple syntax, it was realized that a value type generally performed better and so `System.ValueTuple<...>` was introduced, effectively replacing `System.Tuple<...>` in all cases except for backward compatibility with existing APIs that depend on `System.Tuple<...>`.

Wrapping Up

What many folks didn't realize when it was first introduced is that the new C# 7.0 tuple all but replaces anonymous types—and provides additional functionality. Tuples can be returned from methods, for example, and the item names are persisted in the API such that meaningful names can be used in place of `ItemX` type naming. And, like anonymous types, tuples can even represent complex hierarchical structures such as those that might be constructed in more complex LINQ queries (albeit, like with anonymous types, developers should do this with caution). That said, this could possibly lead to situations where the tuple value type exceeds 128 bytes and, therefore, might be a corner case for when to use anonymous types because it's a reference type. Except for these corner cases (accessing via typical reflection might be another example), there's little to no reason to use an anonymous type when programming with C# 7.0 or later.

The ability to program with a tuple type object has been around for a long time (as mentioned, a tuple class, `System.Tuple<...>`, was introduced with the .NET Framework 4, but was available in Silverlight before that). However, these solutions never had an accompanying C# syntax, but rather nothing more than a .NET API. C# 7.0 brings a first-class tuple syntax that enables literals—like `var tuple = (42, "Inigo Montoya")`—implicit typing, strong typing, public API utilization, integrated IDE support for named `ItemX` data and more. Admittedly, it might not be something you use in every C# file, but it's likely something you'll be grateful to have when the need arises and you'll welcome the tuple syntax over the alternative out parameter or anonymous type.

Much of this article derives from my "Essential C#" book (IntelliTect.com/EssentialCSharp), which I'm currently in the midst of updating to "Essential C# 7.0." For more information on this topic, check out Chapter 3. ■

MARK MICHAELIS is founder of *IntelliTect*, where he serves as its chief technical architect and trainer. For nearly two decades he's 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](https://twitter.com/markmichaelis) or via e-mail at mark@IntelliTect.com.

THANKS to the following Microsoft technical expert for reviewing this article:
Mads Torgersen

TUPLE ITEM NAMING GUIDELINES

- Do use camelCase for all variables declared using tuple syntax.
- Consider using PascalCase for all tuple item names.

2017
Orlando

ROYAL PACIFIC RESORT
AT UNIVERSAL ORLANDO
NOVEMBER 12-17

Visual Studio **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Coding in Paradise

Grab your flip flops, and your laptops, and make plans to attend Visual Studio Live! (VSLive!™), the conference more developers rely on to expand their .NET skills and the ability to build better applications.

Over six full days of unbiased and cutting-edge education on the Microsoft Platform, developers, engineers, designers, programmers and more will soak in the knowledge on everything from Visual Studio and the .NET framework, to AngularJS, ASP.NET and Xamarin.



CONNECT WITH LIVE! 360



twitter.com/live360
[@live360](https://twitter.com/live360)



facebook.com
Search "Live 360"



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

EVENT PARTNERS



PLATINUM SPONSOR



SUPPORTED BY





TECH EVENTS WITH PERSPECTIVE

5 GREAT CONFERENCES, 1 GREAT PRICE

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



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

TURN THE PAGE FOR MORE EVENT DETAILS →

NEW: HANDS-ON LABS



Join us for full-day, pre-conference hands-on labs Sunday, November 12.

Only \$595 through August 11

Whether you are an

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

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

REGISTER BY AUGUST 11 AND SAVE \$500!*

Use promo code ORLAUG4

*Savings based on 5-day packages only. See website for details.

**REGISTER
NOW**



ALM / DEVOPS		CLOUD COMPUTING		NATIVE CLIENT		SOFTWARE PRACTICES		VISUAL STUDIO / .NET FRAMEWORK		WEB CLIENT	
NEW Full Day Hands-On Labs: Sunday, November 12, 2017											
9:00 AM	6:00 PM	VSS01 Full Day Hands-On Lab: Busy Developer's HOL on Angular - <i>Ted Neward</i>						VSS02 Full Day Hands-On Lab: From 0-60 in a day with			
Pre-Conference Workshops: Monday, November 13, 2017											
8:30 AM	5:30 PM	VSM01 Workshop: Distributed Cross-Platform Application Architecture - <i>Jason Bock & Rockford Lhotka</i>				VSM02 Workshop: Artificial Intelligence or DevOps?? - <i>Brian Randell</i>				VSM03 Workshop: Designing, Developing,	
6:30 PM	8:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:30pm - Meet at Conference Registration Desk to walk over with the group									
Day 1: Tuesday, November 14, 2017											
8:00 AM	9:00 AM	Visual Studio Live! KEYNOTE: To Be Announced									
9:15 AM	10:30 AM	VST01 Front-end Web Development in 2017 for the Front-endally Challenged Microsoft Developer - <i>Chris Klug</i>				VST02 Go Mobile with C#, Visual Studio, and Xamarin - <i>James Montemagno</i>				VST03 Microservices with Azure Container Service & Service Fabric - <i>Vishwas Lele</i>	
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - <i>Pacifica 7</i>									
11:00 AM	12:00 PM	LIVE! 360 KEYNOTE: To Be Announced - <i>Pacifica 6</i>									
12:00 PM	12:45 PM	Lunch • Visit the EXPO									
12:45 PM	1:30 PM	Dessert Break • Visit the EXPO									
1:30 PM	2:45 PM	VST05 ASP.NET Core MVC—What You Need to Know - <i>Phillip Japikse</i>				VST06 Optimizing and Extending Xamarin.Forms Mobile Apps - <i>James Montemagno</i>				VST07 Tactical DevOps with Visual Studio Team Services - <i>Brian Randell</i>	
2:45 PM	3:15 PM	Networking Break • Visit the EXPO - <i>Pacifica 7</i>									
3:15 PM	4:30 PM	VST09 Angular(2)—The 75-Minute Crash Course - <i>Chris Klug</i>				VST10 Building a Cross-Platform Mobile App Backend in the Cloud - <i>Nick Landry</i>				VST11 Database Lifecycle Management and the SQL Server Database - <i>Brian Randell</i>	
4:40 PM	5:00 PM	VST13 Fast Focus: Aurelia vs. Just Angular - <i>Chris Klug</i>				VST14 Fast Focus: Tips & Tricks for Xamarin Development - <i>James Montemagno</i>				VST15 Fast Focus on Azure Functions - <i>Rachel Appel</i>	
5:10 PM	5:30 PM	VST17 Fast Focus: Web Security 101 - <i>Brock Allen</i>				VST18 Fast Focus: Cross-Platform Code Reuse - <i>Rockford Lhotka</i>				VST19 Fast Focus: Exploring Microservices in a Microsoft Landscape - <i>Marcel de Vries</i>	
5:30 PM	7:30 PM	Exhibitor Reception - <i>Pacifica 7</i>									
Day 2: Wednesday, November 15, 2017											
8:00 AM	9:15 AM	VSW01 User Authentication for ASP.NET Core MVC Applications - <i>Brock Allen</i>				VSW02 Cloud Oriented Programming - <i>Vishwas Lele</i>				VSW03 Overcoming the Challenges of Mobile Development in the Enterprise - <i>Roy Cornelissen</i>	
9:30 AM	10:45 AM	VSW05 Building AngularJS Component-Based Applications - <i>Miguel Castro</i>				VSW06 Building Modern Web Apps with Azure - <i>Eric D. Boyd</i>				VSW07 Creating a Release Pipeline with Team Services - <i>Esteban Garcia</i>	
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - <i>Pacifica 7</i>									
11:30 AM	12:30 PM	LIVE! 360 KEYNOTE: To Be Announced - <i>Pacifica 6</i>									
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch									
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO									
2:00 PM	3:15 PM	VSW09 Securing Web APIs in ASP.NET Core - <i>Brock Allen</i>				VSW10 A/B Testing, Canary Releases and Dark Launching, Implementing Continuous Delivery on Azure - <i>Marcel de Vries</i>				VSW11 The Zen of UI Automation Testing - <i>Rachel Appel</i>	
3:15 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - <i>Pacifica 7</i>									
4:00 PM	5:15 PM	VSW13 Build Object-Oriented Enterprise Apps in JavaScript with TypeScript - <i>Rachel Appel</i>				VSW14 Lock the Doors, Secure the Valuables, and Set the Alarm - <i>Eric D. Boyd</i>				VSW15 Unit Testing & Test-Driven Development (TDD) for Mere Mortals - <i>Benjamin Day</i>	
8:00 PM	10:00 PM	Live! 360 Dessert Luau - <i>Wantilan Pavilion</i>									
Day 3: Thursday, November 16, 2017											
8:00 AM	9:15 AM	VSH01 HTTP/2: What You Need to Know - <i>Robert Boedigheimer</i>				VSH02 PowerApps and Flow Part II: Package, Embed, and Extend Your Applications - <i>Manas Maheshwari & Pratap Ladhani</i>				VSH03 Exploring C# 7 New Features - <i>Adam Tuliper</i>	
9:30 AM	10:45 AM	VSH05 ASP.NET Tag Helpers - <i>Robert Boedigheimer</i>				VSH06 Storyboarding 101 - <i>Billy Hollis</i>				VSH07 .NET Standard—From Noob to Ninja - <i>Adam Tuliper</i>	
11:00 AM	12:00 PM	Visual Studio Live! Panel: To Be Announced - <i>Brian Randell (Moderator), Damian Brady, Jeremy Clark, Esteban Garcia, Billy Hollis, & Adam Tuliper</i>									
12:00 PM	1:00 PM	Lunch on the Lanai - <i>Lanai / Pacifica 7</i>									
1:00 PM	2:15 PM	VSH09 I See You: Watching the User with Reactive Forms - <i>Deborah Kurata</i>				VSH10 Continuous Integration and Deployment for Mobile Using Azure Services - <i>Kevin Ford</i>				VSH11 Deploying Straight to Production: A Guide to the Holy Grail - <i>Damian Brady</i>	
2:30 PM	3:45 PM	VSH13 Angular Routing - <i>Deborah Kurata</i>				VSH14 XAML Inception—Deep Composition for Better UI - <i>Billy Hollis</i>				VSH15 Application Insights: Measure Your Way to Success - <i>Esteban Garcia</i>	
4:00 PM	5:00 PM	Next? Visual Studio Live! Networking Event - <i>Brian Randell (Moderator), Damian Brady, Jeremy Clark, Esteban Garcia, Billy Hollis, & Deborah Kurata</i>									
Post-Conference Workshops: Friday, November 17, 2017											
8:00 AM	5:00 PM	VSF01 Workshop: Angular Fundamentals - <i>John Papa</i>						VSF02 Workshop: Building, Running & Microservices with Docker Containers on Azure			

Presented in Partnership with
Magenic

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT



ROYAL PACIFIC RESORT
AT UNIVERSAL ORLANDO
NOVEMBER 12-17

WEB SERVER

MODERN APPS LIVE!

Check Out These Additional Sessions for Developers at Live! 360

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING



Office & SharePoint Live! features 15+ developer sessions, including:

- **NEW!** Full Day Hands-On Lab: Developing Extensions for Microsoft Teams - *Paul Schaefflein*
- Workshop: Mastering the SharePoint Framework - *Andrew Connell*
- TypeScript for SharePoint Developers - *Rob Windsor*
- Building Office Add-ins for Outlook with Angular - *Andrew Connell*
- Developing SharePoint Framework Components Using Visual Studio - *Paul Schaefflein*
- What Every Developer Needs to Know about SharePoint Development Online or On-Prem - *Robert Bogue*
- Build a Complete Business Solution Using Microsoft Graph API through Client Side Web Parts - *Julie Turner*

SQL Server LIVE!

TRAINING FOR DBAS AND IT PROS



SQL Server Live! features 30+ developer sessions, including:

- **NEW!** Full Day Hands-On Lab: Developer Dive into SQL Server 2016
- Turbo Boost - SQL Tricks Everybody MUST Know - *Pinal Dave*
- Advanced SSIS Package Authoring with Biml - *Tim Mitchell*
- Graph DB Support in SQL Server 2017 - *Karen Lopez*
- Big Data Technologies: What, Where and How to Run Them on Azure - *Andrew Brust*
- Top Five SQL Server Query Tuning Tips - *Janis Griffin*
- Workshop: Big Data, BI, and Analytics on The Microsoft Stack - *Andrew Brust*

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS



TechMentor features 20 + developer sessions, including:

- **NEW!** Full Day Hands-On Lab: Ethical Hacking with Kali Linux - *Mike Danseglio & Avril Salter*
- Workshop: Windows Security—How I Do It! - *Sami Laiho*
- Hardware, Camtasia, and a Storyline: Creating Your Own User Training - *Greg Shields*
- Make Your PowerShell Scripts Bulletproof with Pester - *Melissa Januszko*
- Controlling Your Azure Spend - *Timothy Warner*
- PowerShell Scripting Secrets - *Jeffery Hicks*
- In-Depth Introduction to Docker - *Neil Peterson*



VSLIVE.COM/ORLANDOMSDN

Xamarin and Xamarin.Forms - *Roy Cornelissen*

Service Oriented Technologies: & Implementing WCF and the Web API - *Miguel Castro*

Pre-Con Workshops: Monday, Nov. 13

MAM01 Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*

Dine-A-Round Dinner

Day 1: Tuesday, November 14, 2017

Modern Apps Live! KEYNOTE PANEL: Industry Trends, Technology, and Your Career - *Matt Lockhart (Moderator)*

MAT01 Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*

Networking Break • Visit the EXPO - *Pacifica 7*

LIVE! 360 KEYNOTE

Lunch • Visit the EXPO

Dessert Break • Visit the EXPO

MAT02 Architecture: The Key to Modern App Success - *Brent Edwards*

Networking Break • Visit the EXPO - *Pacifica 7*

MAT03 Modern Mobile Development: Build a Single App For iOS, Android, and Windows with Xamarin Forms - *Kevin Ford*

MAT04 Fast Focus: Hybrid Web Frameworks - *Allen Conway*

MAT05 Fast Focus: Web Assembly - *Jason Bock*

Exhibitor Reception - *Pacifica 7*

Day 2: Wednesday, November 15, 2017

MAW01 Focus on the User Experience #FTW - *Jim Barrett*

MAW02 DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*

Networking Break • Visit the EXPO - *Pacifica 7*

LIVE! 360 KEYNOTE

Birds-of-a-Feather Lunch

Dessert Break • Visit the EXPO

MAW03 Security with Speed for Modern Developers - *Michael Lester*

Networking Break • Visit the EXPO • Raffle @ 3:30 p.m.

MAW04 Coding for Quality and Maintainability - *Jason Bock*

Live! 360 Dessert Luau - *Wantilan Pavilion*

Day 3: Thursday, November 16, 2017

MAH01 Modern Web Development: Building Server Side Using ASP.NET Core, MVC, Web API, and Azure - *Allen Conway*

MAH02 Modern Web Development: Building Client Side Using TypeScript and Angular - *Allen Conway*

Modern Apps Live! Panel: Mobile Development Technologies - *Rockford Lhotka (Moderator), James Montemagno, Kevin Ford*

Lunch on the Lanai - *Lanai / Pacifica 7*

MAH03 Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randall*

MAH04 Using All That Data: Power BI to the Rescue - *Scott Diehl*

Next? Modern Apps Live! Networking Event - *Rockford Lhotka (Moderator)*

Post-Con Workshops: Friday, Nov. 17

MAF01 Workshop: Modern App Deep Dive—Xamarin, Responsive Web, UWP - *Kevin Ford, Brent Edwards, Allen Conway*

VST04 What's New in Visual Studio 2017 - *Robert Green*

VST08 To Be Announced

VST12 Top 10 Entity Framework Core Features Every Developer Should Know - *Philip Japikse*

VST16 Fast Focus: Busting .NET Myths - *Jason Bock*

VST20 Fast Focus: Dependency Injection in 20 Minutes - *Miguel Castro*

VSW04 Building Apps with Microsoft Graph and Visual Studio - *Robert Green*

VSW08 Bots are the New Apps: Building Bots with ASP.NET Web API & Language Understanding - *Nick Landry*

VSW12 To Be Announced

VSW16 PowerApps, Flow, and Common Data Service: Empowering Businesses with the Microsoft Business Application Platform - *Charles Sterling*

VSH04 Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*

VSH08 Devs vs. Ops: Making Friends with the Enemy - *Damian Brady*

VSH12 Design Patterns: Not Just for Architects - *Jeremy Clark*

VSH16 DI Why? Getting a Grip on Dependency Injection - *Jeremy Clark*

Continuously Deploying - *Marcel de Vries & Rene van Osnabrugge*



Salt and Pepper

We're currently celebrating the 50th anniversary of the classic Beatles album "Sgt. Pepper's Lonely Hearts Club Band" (SPLHCB). Its beautiful strains brighten my office atmosphere as I write these words. Other writers may address its groundbreaking musical effects (see the interview with the recording engineer at bit.ly/2rEetU0), or its place in the evolution of rock music. Or its spoofs, from Doonesbury (bit.ly/2sjceHR) to National Lampoon (bit.ly/2sHvCAN). But contemplating Sgt. Pepper today makes me notice the ways in which changes in listening technology have driven changes in musical artistry.

At the time of SPLHCB's release, essentially all music was sold on LP albums. You had to buy the whole package, and listen to all of its songs sequentially. The progression to cassette tapes and then to CDs didn't change that constraint. Therefore, the artist had to carefully compose the sequence of songs on the album, as their influence on each other was inescapable. The Beatles placed George Harrison's introspective, sitar-laden "Within You, Without You" ahead of Paul McCartney's whimsical "When I'm Sixty-Four," driven by its trio of clarinets. Reversing that order would have induced entirely different feelings in even a casual listener. They carefully slotted Ringo's "With a Little Help From My Friends" into the second track, where it would do the least damage, and gave it introductory applause effects to pre-dispose the audience's perceptions toward approval.

The digital revolution—the liberation of pure thought-stuff from the profane physical medium on which it resided—undid these artistic decisions. Online stores such as iTunes and Amazon sold individual songs, so you didn't have to buy the bad ones. Any listener could easily rip CDs to disk, composing playlists that mixed and matched tracks and artists in any order. We lost that part of the artist's intention.

And that liberation/loss doesn't solely affect the album's song sequences. It also ripples through the content of individual songs. An artist releasing an album today can't know which track the listener is hearing before or after any song. Therefore, each song needs to be an island unto itself, rather than part of an artistic whole. How can anyone compose or play the final orchestral crescendo in "A Day in the Life" (bit.ly/1Lln4Z), terminating in the world's most famous piano chord, without intending to signal the end of the larger work to which it belongs? (That remains my biggest dilemma on playing Sgt. Pepper from end to end: What the heck do I play next?)

Sometimes this liberation from pre-imposed order is good. When I search Spotify for an artist, it will by default play a shuffle of that artist's most popular tunes. If I'm introducing my daughters

to the Grateful Dead, that's not such a bad thing. But sometimes it's not so good: allowing them to reach the age of majority without experiencing the spare, wandering piano at the end of Bruce Springsteen's "Incident on 57th Street" segueing straight into the first crashing chords of "Rosalita" would be abdicating my duty as a parent. They ask me, "Daddy, what was all the fuss about the Beatles' White Playlist?" and I'm not sure what to tell them. Some things are worth digging for.

The digital revolution—the liberation of pure thought-stuff from the profane physical medium on which it resided—undid these artistic decisions.

I can hear you thinking: Plattski, you always were a Luddite in this industry, failing to worship technology for its own sake as we all do, insisting on a practical benefit before you'd jump on any bandwagon (see, for example, my March 2016 column, "The Internet of Invisible Things," msdn.com/magazine/mt683803). But now you're going positively Amish on us. Your nostalgia for the original artistic sequences is like an old jeweler moaning over the loss of those beautiful metal components in a mechanical watch, when a simple quartz oscillator keeps far better time for a tenth of the price.

I'm not saying you shouldn't make your own playlists, I certainly make mine. (Try my Trop Rock playlist that Spotify automatically exports to my Facebook page.) But the 50th anniversary of Sgt. Pepper reminds me to carefully examine the original artists' chosen sequence and content, especially for albums which pre-date easy ripping and self-composition. I expect some cheers for this idea now, and even more in two years, when we celebrate the 50th anniversary of "Abbey Road." ■

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.



Looking for a powerful distributed cache?

Get the easiest, most powerful in-memory data grid designed to scale your .NET applications.

- ▶ Fast and linearly scalable
- ▶ Enterprise-grade availability
- ▶ Industry-leading ease of use
- ▶ Integrated in-memory computing

Go with the industry leader in distributed caching

Trusted by hundreds of enterprise customers for more than 12 years, ScaleOut's distributed caching technology delivers rock-solid performance, legendary ease of use, and world-class support. Unlike Redis, it offers a better migration path from AppFabric Caching. Here's why:

	ScaleOut	Redis
Source-code compatible AppFabric Caching APIs for seamless migration	✓	✗
Architected from the ground up for automatic scaling and high availability	✓	✗
Fully coherent data storage and access for mission-critical applications	✓	✗
Advanced .NET features: distributed LINQ query, C# MapReduce, and more	✓	✗

Replacing AppFabric Caching? Trouble switching to Redis?

Check out our AppFabric-compatible drop-in: www.scaleoutsoftware.com/appfabric





Build incredible software using 800+ controls,
the unique Syncfusion Data Platform,
and more

★ No per-user charges ★ No per-developer charges ★ No per-server charges

GET STARTED FOR FREE!

www.syncfusion.com/MSDNunlimited

 Syncfusion®



Powerful File APIs that are easy and intuitive to use
Native APIs for .NET, Java & Cloud

Using Aspose.Words for .NET to
Convert Word Docs to HTML -
Case Study

Adding File Conversion and
Manipulation to Business Systems

DOC, XLS, JPG,
PNG, PDF, BMP,
MSG, PPT, VSD,
XPS & many other
formats.

 www.aspose.com

EU Sales: +44 141 628 8900

US Sales: +1 903 306 1676
sales@asposeytd.com

AU Sales: +61 2 8006 6987



Aspose.Total

Every Aspose API combined in one powerful suite.

➤ Aspose.Cells

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

➤ Aspose.BarCode

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

➤ Aspose.Words

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

➤ Aspose.Tasks

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

➤ Aspose.Pdf

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

➤ Aspose.Email

MSG, EML, PST, MHT, OST
OFT...

➤ Aspose.Slides

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

➤ Aspose.Imaging

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

and many more!



Contact Us:

US: +1 903 306 1676

EU: +44 141 628 8900

AU: +61 2 8006 6987

sales@asposeptyltd.com



Working with Files?

Try Aspose File APIs

Convert
Print
Create
Combine
Modify



files from your applications!

Over 15,000 Happy Customers

.NET

Java

Cloud

Get your FREE evaluation copy at www.aspose.com

Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

ASPOSE.CELLS IS A PROGRAMMING API that allows developers to create, manipulate and convert Microsoft Excel spreadsheet files from within their own applications. Its powerful features make it easy to convert worksheets and charts to graphics or save reports to PDF.

Aspose.Cells speeds up working with Microsoft Excel files. The

A flexible API for simple and complex spreadsheet programming.

API is a flexible tool for simple tasks such as file conversion, as well as complex tasks like building models. Developers control page layout, formatting, charts and formulas. They can read and write spreadsheet files and save out to a wide variety of image and text file formats.

Fast, scalable, and reliable, Aspose.Cells saves time and effort compared to using Microsoft Office

	A	B	C	D
1	TeXT	FORMULA TEXT	FORMULA AND RESULT	DATA
2	ASC	ASC("EXCEL")	EXCEL	
3	CHAR	CHAR(65)	A	
4	CODE	CODE("A")		65
5		CODE("DEF")		68

Aspose.Cells lets developers work with data sources, formatting, even formulas.

Automation.

Common Uses

- Building dynamic reports on the fly.
- Creating Excel dashboards with charts and pivot tables.
- Rendering and printing spreadsheets and graphics with high fidelity.
- Exporting data to, or importing from, Excel and other spreadsheets.
- Generating, manipulating and editing spreadsheets.
- Converting spreadsheets to images or other file formats.

Key Features

- A complete spreadsheet manipulation solution.
- Flexible data visualization and

reporting.

- Powerful formula engine.
- Complete formatting control.

Supported File Formats

XLS, XLSX, XLSM, XMPS, XLTX, XLTM, ODS, XPS, SpreadsheetML, tab delim., CSV, TXT, PDF, HTML, and many image formats including SVG, TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms



Pricing Info

	Standard	Enhanced		Standard	Enhanced
Developer Small Business	\$999	\$1498	Site Small Business	\$4995	\$7490
Developer OEM	\$2997	\$4494	Site OEM	\$13986	\$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposeytd.com

Oceania: +61 2 8006 6987

Aspose.Cells for

.NET, Java, Cloud & more

File Formats

XLS, CSV, ODS, PDF, SVG, HTML, PNG, BMP, XPS, JPG SpreadsheetML and many others.

Spreadsheet Manipulation

Aspose.Cells lets you create, import, and export spreadsheets and also allows you to manipulate contents, cell formatting, and file protection.

Creating Charts

Aspose.Cells comes with complete support for charting and supports all standard chart types. Also, you can convert charts to images.

Graphics Capabilities

Easily convert worksheets to images as well as adding images to worksheets at runtime.

100% Standalone

Aspose.Cells does not require Microsoft Office to be installed on the machine in order to work.

Get your FREE Trial at
<http://www.aspose.com>

Aspose.Words

Program with word processing documents independently of Microsoft Word

- Solution for document creation, manipulation and conversion.
- Advanced mail merge functionality.

ASPOSE.WORDS IS AN ADVANCED PROGRAMMING API

that lets developers perform a wide range of document processing tasks with their own applications. Aspose.Words makes it possible to generate, modify, convert, render and print documents without Microsoft Word. It provides sophisticated and flexible access to, and control over, Microsoft Word files.

Aspose.Words is powerful, user-friendly and feature rich. It saves

developers time and effort compared to using Microsoft Office Automation and makes gives them powerful document management tools.

Aspose.Words makes creating, changing and converting DOC and other word processing file formats fast and easy.

Generate, modify, convert, render and print documents without Microsoft Word.

	Table			
	Column 1	Column 2	Column 3	Column 4
Row 1	Cell 1	Cell 2	Cell 3	Cell 4
Row 2	Cell 1	Cell 2	Cell 3	
Row 3	Cell 1	Cell 2		

Aspose.Words has sophisticated controls for formatting and managing tables and other content.

Common Uses

- Generating reports with complex mail merging; mail merging images.
- Populating tables and documents with data from a database.
- Inserting formatted text, paragraphs, tables and images into Microsoft Word documents.
- Adding barcodes to documents.
- Inserting diagrams and watermarks into Word documents.
- Formatting date and numeric fields.

Key Features

- A complete Microsoft Word document manipulation solution.
- Extensive mail merge features.
- Complete formatting control.
- High-fidelity conversion, rendering and printing.

Supported File Formats

DOC, DOCX, ODT, OOXML, XML, HTML, XHTML, MHTML, EPUB, PDF, XPS, RTF, and a number of image formats, including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms



	Pricing Info				
	Standard	Enhanced	Standard	Enhanced	
Developer Small Business	\$999	\$1498	Site Small Business	\$4995	\$7490
Developer OEM	\$2997	\$4494	Site OEM	\$13986	\$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposeytd.com

Oceania: +61 2 8006 6987

Case Study: Aspose.Words for .NET

ProHire Staffing - Using Aspose.Words for .NET to convert Word Docs to HTML

PROHIRE IS THE WORKFORCE SOLUTIONS LEADER IN THE UNITED STATES AND SPECIALIZE IN THE RECRUITMENT OF SALES AND SALES MANGEMENT PROFESSIONALS.

We were founded with the goal of becoming the premier provider of executive search and placement services to the Fortune 500 and Inc. 500 Companies.

Problem

ProHire uses Bullhorn ATS as its Application Tracking System to track the electronic handling of its recruitment needs. We wanted to integrate the Bullhorn API with our new website. Our goal was to convert MS Word Documents resumes into a clean and concise HTML format into our existing .Net Stack. The converted HTML resume version needed to look close to the original.

Looking for a Solution

We chose the ASPOSE.Words product because it easily integrated into our existing .Net stack, and provided a quality MS Word to HTML conversion. The product was easy to download, and with a few lines of code we were up and running. We found the primary

difference between the Aspose.Words and other products was the obvious conversion quality from MS Word to HTML.

Finding a Solution

We had tested other products that converted Word to HTML. Every one we tested had some problem with the conversion. Some of them lost

elements of the resume during the conversion. Most of them changed the format of the resume or changed the color of the text unexpectedly. This is unacceptable when you are sending a resume to a hiring manger. We were very satisfied with the results. We did not need

any technical support because documentation was sufficient to us.

Implementation

Once we had the Aspose DLL our developer was able to implement Aspose.Words for .NET in a few hours. The transitions with Aspose.Words for .NET was very painless to do.

Outcome

We are very pleased with the success of our Aspose.Words for .NET implementation. Aspose.Words is a very powerful development tool that is well documented and easy to install. The documentation is easy to understand and use. If you want a product to convert Word Docs to HTML look no further. ProHire is happy to recommend Aspose.

This is an extract from a case study on our website. For the full version, go to: www.aspose.com/corporate/customers/case-studies.aspx

"The transitions with Aspose.Words for .NET was very painless to do."



The converted HTML resume version needed to look close to the original.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposeptyltd.com

Oceania: +61 2 8006 6987



Open, Create, Convert, Print & Save Files

from within your *own* applications.

> ASPOSE.TOTAL

allows you to process these file formats:

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



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



ASPOSE

File Format APIs

Contact Us:

US: +1 903 306 1676

EU: +44 141 628 8900

AU: +61 2 8006 6987

sales@asposeptyltd.com

Helped over 11,000 companies and over 300,000 users work with documents in their applications.

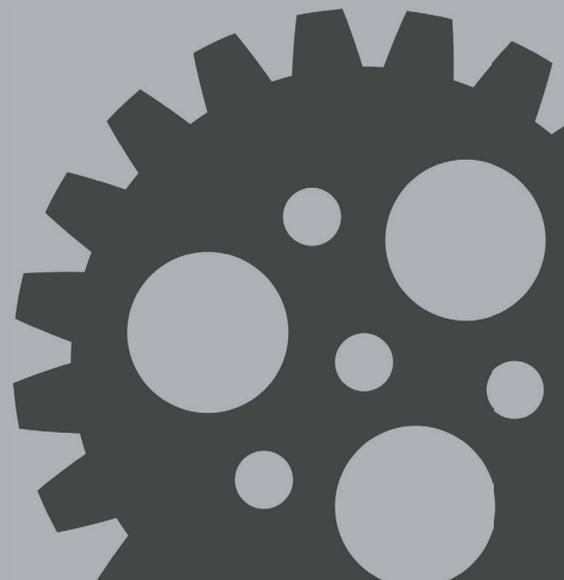
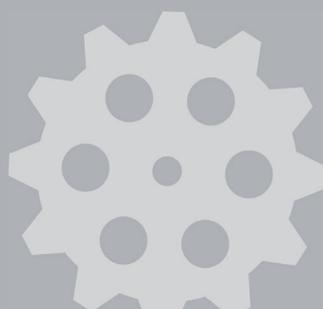
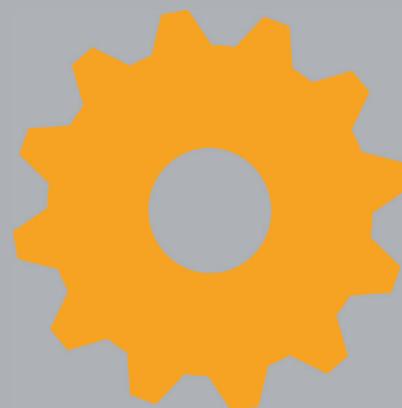


File Format APIs



GET STARTED NOW

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



Adding File Conversion and Manipulation to Business Systems

How often do people in your organization complain that they can't get information in the file format and layout they want? Converting documents from one format to another without losing layout and formatting should be simple, but it can be frustrating for both users and developers.

EXTRACTING DATA FROM A DATABASE AND DELIVERING IT TO THE SALES TEAM AS A REPORT, complete with charts and corporate branding, is fine. Until the sales team says that they want it as a Microsoft Excel file, and could you add a dashboard?

Using information from online forms in letters that can be printed and posted is easy. But what if you also want to add tracking barcodes and archive a digital copy as a PDF?

Ensuring that your business system supports all the different Microsoft Office file formats your users want can be difficult. Sometimes the native file format support of your system lets you down. When that is the case, use tools that extend that capability. A good tool can save you time and effort.

Document Conversion Options

Building your own solution: Time-consuming and costly, this option is only sensible if the solution you develop is central to your business.

Using Microsoft Office Automation: Microsoft Office

Automation lets you use Microsoft Office programs server-side. It is not how the Office products were designed to be used. It can work well but you might notice issues with the stability, security and speed of the system, as well as cost.

Using an API: The API market has lots of free and commercial solutions, some very focused, some feature-rich. An API integrates with your code and gives you access to a range of new features.

barcodes and OCR. The APIs are optimised for stability, speed and ease of use. Our APIs save users weeks, sometimes months, of effort.

Aspose creates APIs that work independently of Microsoft Office Automation.



Finding the Right Tool

To find the product that's right for you, take a systematic approach:

- List must-have and nice-to-have features.
- Research the market.
- Ask for recommendations.
- Select a few candidates .
- Run trials.
- Evaluate
 - ease of use,
 - support and documentation,
 - performance, and
 - current and future needs.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposeptyltd.com

Oceania: +61 2 8006 6987

Aspose.BarCode

A complete toolkit for barcode generation and recognition

- Generate barcodes with customer defined size and color.
- Recognize a large number of barcode types from images.

ASPOSE.BARCODE IS A ROBUST AND RELIABLE BARCODE GENERATION AND RECOGNITION API that allows developers to add barcode generation and recognition functionality to their applications quickly and easily.

Aspose.BarCode supports most established barcode specifications. It can export generated barcodes to multiple image formats, including BMP, GIF, JPED, PNG and TIFF.

Aspose.BarCode gives you full control over every aspect of the barcode

Robust and reliable barcode generation and recognition.

image, from background and bar color, through image quality, rotation angle, X-dimension, captions, and resolution.

Aspose.BarCode can read and recognize most common 1D and 2D barcodes from any image and at any angle. Filters help developers



Aspose.BarCode offers a large number of symbologies and formatting options.

clean up difficult to read images to improve recognition.

Common Uses

- Generating and recognizing barcode images.
- Printing barcode labels.
- Enhancing workflow by adding barcode functionality.
- Using recognition functions to drive real-life work processes.

Key Features

- Barcode generation and recognition.
- Comprehensive support for 1D and 2D symbologies.
- Image processing for improved recognition.

Supported File Formats

JPG, TIFF, PNG, BMP, GIF, EMF, WMF,

EXIP and ICON.

Format support varies across platforms.

Supported Barcodes

Linear: EAN13, EAN8, UPCA, UPCE, Interleaved2of5, Standard2of5, MSI, Code11, Codabar, EAN14(SCC14), SSCC18, ITF14, Matrix 2 of 5, PZN, Code128, Code39 Extended, Code39 Standard, OPC, Code93 Extended, Code93 Standard, IATA 2 of 5, GS1Code128, ISBN, ISMN, ISSN, ITF6, Pharmacode, DatabarOmniDirectional, VIN, DatabarTruncated, DatabarLimited, DatabarExpanded, PatchCode, Supplement 2D: PDF417, MacroPDF417, DataMatrix, Aztec, QR, Italian Post 25, Code16K, GS1DataMatrix **Postal:** Postnet, Planet, USPS OneCode, Australia Post, Deutsche Post Identcode, AustralianPosteParcel, Deutsche Post Leticode, RM4SCC, SingaporePost, SwissPostParcel

Platforms



	Pricing Info				
	Standard	Enhanced	Standard	Enhanced	
Developer Small Business	\$599	\$1098	Site Small Business	\$2995	\$5490
Developer OEM	\$1797	\$3294	Site OEM	\$8386	\$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposeptyltd.com

Oceania: +61 2 8006 6987



Aspose for Cloud

The easiest API to Create, Convert & Automate Documents in the cloud.



**Convert
Create
Render
Combine
Modify**

without installing anything!

<p>Aspose.Words for Cloud </p> <p>Create and convert docs Manipulate text Render documents Annotate</p>	<p>Aspose.Cells for Cloud </p> <p>Create spreadsheets Convert spreadsheets Manipulate cells and formulas Render spreadsheets</p>
<p>Aspose.Slides for Cloud </p> <p>Create presentations Manage slides Edit text and images Read and convert</p>	<p>Aspose.Pdf for Cloud </p> <p>Create and convert PDFs Manipulate text, images Add pages, split, encrypt Manage stamps</p>
<p>Aspose.Email for Cloud </p> <p>Create, update, and convert messages Extract attachments Use with any language</p>	<p>Aspose.BarCode for Cloud </p> <p>Generate barcodes Read barcodes Set attributes Multiple image formats</p>

Free Evaluation at www.aspose.com

• +44 141 628 8900

• +1 903 306 1676
sales@asposeptyltd.com

• +61 2 8006 6987

Aspose.Email

Work with emails and calendars without Microsoft Outlook

- Complete email processing solution.
- Message file format support.

ASPOSE.EMAIL IS AN EMAIL PROGRAMMING API

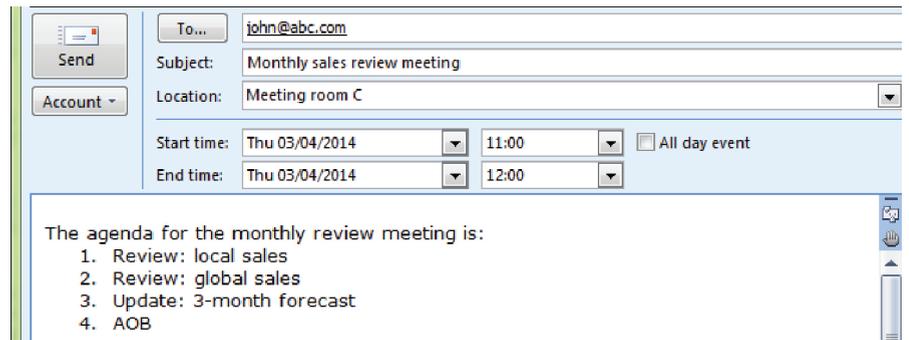
that allows developers to access and work with PST, EML, MSG and MHT files. It also offers an advanced API for interacting with enterprise mail systems like Exchange and Gmail.

Aspose.Email can work with HTML and plain text emails, attachments and embedded OLE objects.

It allows developers to work against SMTP, POP, FTP and Microsoft Exchange servers. It supports mail merge and iCalendar features, customized header and body, searching archives and has many other useful features.

Aspose.Email allows developers to focus on managing email without getting into the core of email and network programming. It gives you the controls you need.

Aspose.
Email works
with HTML
and plain
text emails,
attachments
and embedded
OLE objects.



Aspose.Email lets your applications work with emails, attachments, notes and calendars.

Common Uses

- Sending email with HTML formatting and attachments.
- Mail merging and sending mass mail.
- Connecting to POP3 and IMAP mail servers to list and download messages.
- Connecting to Microsoft Exchange Servers to list, download and send messages.
- Create and update tasks using iCalendar.
- Load from and save messages to file or stream (EML, MSG or MHT formats).

Key Features

- A complete email processing solution.
- Support for MSG and PST formats.
- Microsoft Exchange Server support.
- Complete recurrence pattern solution.

Supported File Formats

MSG, MHT, OST, PST, EMLX, TNEF, and EML.

Format support varies across platforms.

Platforms



Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	\$799	\$1298	Site Small Business	\$3995	\$6490
Developer OEM	\$2397	\$3894	Site OEM	\$11186	\$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposey ltd.com

Oceania: +61 2 8006 6987

Aspose.Pdf

Create PDF documents without using Adobe Acrobat

- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

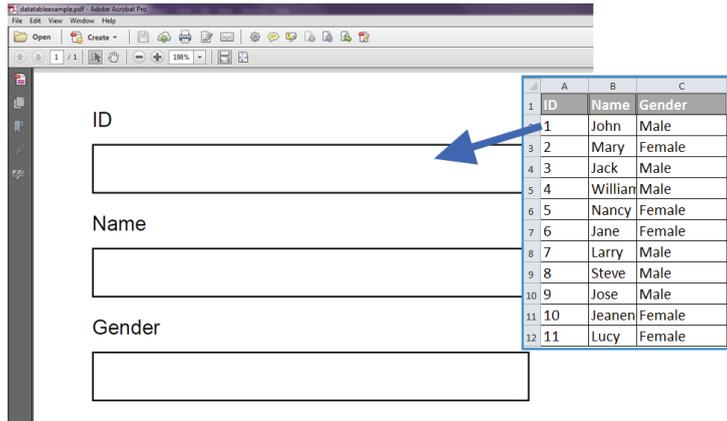
ASPOSE.PDF IS A PDF DOCUMENT CREATION AND MANIPULATION API

that developers use to read, write and manipulate PDF documents without using Adobe Acrobat. Aspose.Pdf is a sophisticated product that integrates with your application to add PDF capabilities.

Aspose.Pdf offers a wealth of features that lets developers compress files, create tables, work with links, add and remove security, handle custom fonts, integrate with external data sources, manage bookmarks, create table of contents, create forms and manage form fields.

Read, write and manipulate PDF documents independently of Adobe Acrobat.

It helps developers add, work with attachments, annotations and PDF form data, add, replace or remove text and images, split, concatenate,



Aspose.Pdf can be used to automatically complete PDF forms with external data.

extract or inset pages, and print PDF documents.

Common Uses

- Creating and editing PDF files.
- Inserting, extracting, appending, concatenating and splitting PDFs.
- Working with text, images, tables, images, headers, and footers.
- Applying security, passwords and signatures.
- Working with forms and form fields.

Key Features

- PDF creation from XML or XSL-FO documents.
- PDF form and field support.
- Advanced security and encryption.
- High-fidelity printing and conversion.
- Supported File Formats
- PDF, PDF/A, PDF/A_1b, PCL, XLS-FO, LaTeX, HTML, XPS, TXT and a range of image formats.

Format support varies across platforms.

Platforms



Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	\$999	\$1498	Site Small Business	\$4495	\$6990
Developer OEM	\$2997	\$4494	Site OEM	\$13986	\$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposepdf.com

Oceania: +61 2 8006 6987

Aspose.Pdf

.Net, Java & Cloud

File Formats

PDF DOC XML XSL-FO XPS HTML BMP JPG PNG
ePUB & other image file formats.

Create and Manipulate PDFs

Create new or edit/manipualte existing PDFs.

Form Field Features

Add form fields to your PDFs. Import and export form fields data from select file formats.

Table Features

Add tables to your PDFs with formatting such as table border style, margin and padding info, column width and spanning options, and more.

Get started today at www.aspose.com

Aspose.Note for .NET

Aspose.Note for .NET is an API that lets developers convert Microsoft OneNote pages to a variety of file formats, and extract the text and document information.

Conversion is fast and high-fidelity. The output looks like the OneNote page, no matter how complex the formatting or layout.

Aspose.Note works independently of Office Automation and does not require Microsoft Office or OneNote to be installed.

Modify, convert, render and extract text and images from Microsoft OneNote files without relying on OneNote or other libraries.

Features

File Formats and Conversion

Microsoft OneNote 2010, 2010 SP1, 2013	Load, Save
PDF	Save
Images (BMP, GIF, JPG, PNG)	Save

Rendering and Printing

Save as Image (BMP, GIF, JPG, PNG)
Save as PDF

Document Management

- Extract text
- Get the number of pages in a document.
- Get page information.
- Extract images.
- Get image information from a document.
- Replace text in document.

Aspose.Imaging

Create Images from scratch.

- Load existing images for editing purposes.
- Render to multiple file formats.

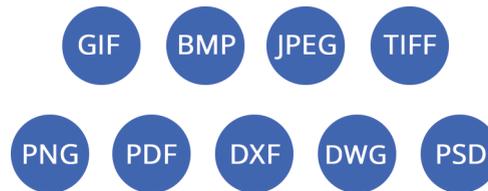
ASPOSE.IMAGING IS A CLASS

LIBRARY that facilitates the developer to create Image files from scratch or load existing ones for editing purpose. Also, Aspose.Imaging provides the means to save the created or edited Image to a variety of formats. All of the above mentioned can be achieved without the need of an Image Editor. It works independent of other applications and although Aspose.Imaging allows you to save to Adobe PhotoShop® format (PSD), you do not need PhotoShop installed on the machine.

Aspose.Imaging is flexible, stable and powerful. It's many features and image processing routines should meet most imaging requirements. Like all Aspose file format components, Aspose.

Imaging introduces support for an advanced set of drawing features along with the core functionality. Developers can

Create images from scratch. or load existing ones...



Aspose.Imaging allows creation and manipulation of images.

draw on Image surface either by manipulating the bitmap information or by using the advanced functionality like Graphics and Paths.

Common Uses

- Create images from scratch.
- Load and Edit existing images.
- Export images to a variety of formats.
- Adding watermark to images.
- Export CAD drawings to PDF & raster image formats.
- Crop, resize & RotateFlip images.
- Extract frames from multipage TIFF image.

Key Features

- Create, edit, and save images
- Multiple file formats
- Drawing features
- Export images

Supported File Formats

BMP, JPG, TIFF, GIF, PNG, PSD, DXF, DWG, and PDF.

Platforms



	Pricing Info			
	Standard	Enhanced	Standard	Enhanced
Developer Small Business	\$399	\$898	Site Small Business	\$1995
Developer OEM	\$1197	\$2694	Site OEM	\$4490
				\$5586
				\$12572

The pricing info above is for .NET.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposeptyltd.com

Oceania: +61 2 8006 6987

Aspose.Slides

Work with presentations without using Microsoft PowerPoint

- Complete solution for working with presentation files.
- Export presentations and slides to portable or image formats.

ASPOSE.SLIDES IS A FLEXIBLE PRESENTATION MANAGEMENT

API that helps developers read, write and manipulate Microsoft PowerPoint documents. Slides and presentations can be saved to PDF, HTML and image file formats without Microsoft PowerPoint.

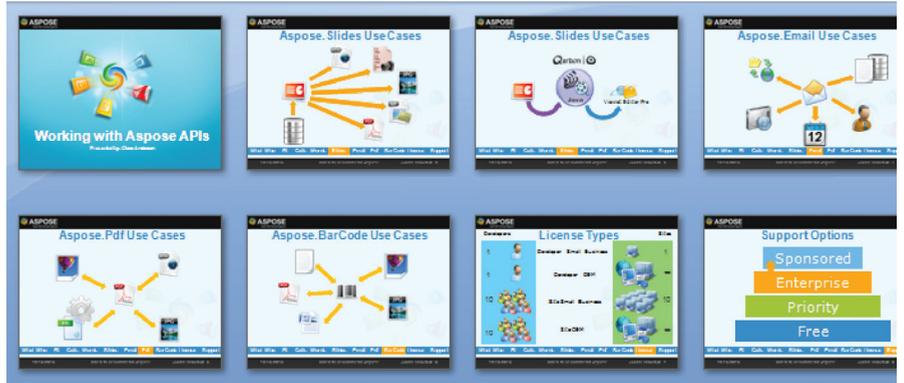
Aspose.Slides offers a number of advanced

features that make it easy to perform tasks such as rendering slides, exporting

Aspose.Slides gives you the tools you need to work with presentation files.

presentations, exporting slides to SVG and printing. Developers use Aspose.Slides to build customizable slide decks, add or remove standard graphics and automatically publish presentations to other formats.

Aspose.Slides gives developers the tools they need to work with presentation files. It integrates quickly and saves time and money.



Aspose.Slides has advanced features for working with every aspect of a presentation.

Common Uses

- Creating new slides and cloning existing slides from templates.
- Handling text and shape formatting.
- Applying and removing protection.
- Exporting presentations to images and PDF.
- Embedding Excel charts as OLE objects.
- Generate presentations from database.

Key Features

- A complete presentation development solution.
- Control over text, formatting and slide elements.
- OLE integration for embedding

external content.

- Wide support for input and output file formats.

Supported File Formats

PPT, HTML, POT, PPS, PPTX, POTX, PPSX, ODP, PresentationML, XPS, PDF and image formats including TIFF and JPG.

Format support varies across platforms.

Platforms



	Pricing Info				
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	\$799	\$1298	Site Small Business	\$3995	\$6490
Developer OEM	\$2397	\$3894	Site OEM	\$11186	\$18172

The pricing info above is for .NET; prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposeptyltd.com

Oceania: +61 2 8006 6987

Support Services

Get the assistance you need, when you need it, from the people who know our products best.

- Free support for all, even when evaluating
- Get the level of support that suits you and your team

NO ONE KNOWS OUR PRODUCTS AS WELL AS WE DO.

We develop them, support them and use them. Our support is handled through our support forums and is available to all Aspose users.

Support

We are developers ourselves and understand how frustrating it is when a technical issue or a quirk in the software stops you from doing what you need to do. This is why we offer free support. Anyone who uses our product, whether they have bought them or are using an evaluation, deserves our full attention and respect. We have four levels of support that can fit your needs.

Everyone who uses Aspose products have access to our free support.



Work with the developers that developed and continue to maintain our products.

Support Options

Free

Everyone who uses Aspose products have access to our free support. Our software developers are on stand-by to help you succeed with your project, from the evaluation to roll-out of your solution.

Priority

If you want to know when you'll hear back from us on an issue and know that your issue is prioritized, Priority Support is for you. It provides a more formal support structure and has its own forum that is monitored by our software engineers.

Enterprise

Enterprise customers often have very specific needs. Our Enterprise Support option gives them access to the product development team and influence over the roadmap. Enterprise Support customers have their own, dedicated issue tracking system.

Sponsored

Available to Enterprise customers that would like to request features, this higher prioritized support can ensure your needed features are on our roadmap. A member of our team will produce a feature specification document to capture your requirements and how we intend to fulfill them so the direction development will take is clear up-front.



Pricing Info

To see the Priority and Enterprise support rates, refer to the product price list, or contact our sales team.

Sponsored Support is unique so pricing is specific to each project. Please contact our sales team to discuss.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@asposeptyltd.com

Oceania: +61 2 8006 6987

We're Here to Help You

Aspose has 4 Support Services to best suit your needs

Free Support

Support Forums with no Charge

Priority Support

24 hour response time in the week,
issue escalation, dedicated forum

Enterprise Support

Communicate with product
managers, influence the roadmap

Sponsored Support

Get the feature you need built now

Technical Support is an issue that Aspose takes very seriously. Software must work quickly and dependably. When problems arise, developers need answers in a hurry. We ensure that our clients receive useful answers and solutions quickly.

[Email](#) • [Live Chat](#) • [Forums](#)

Contact Us

US Sales: +1 903 306 1676

sales@asposeptyltd.com

EU Sales: +44 141 628 8900

AU Sales: +61 2 8006 6987