THE MICROSOFT JOURNAL FOR DEVELOPERS

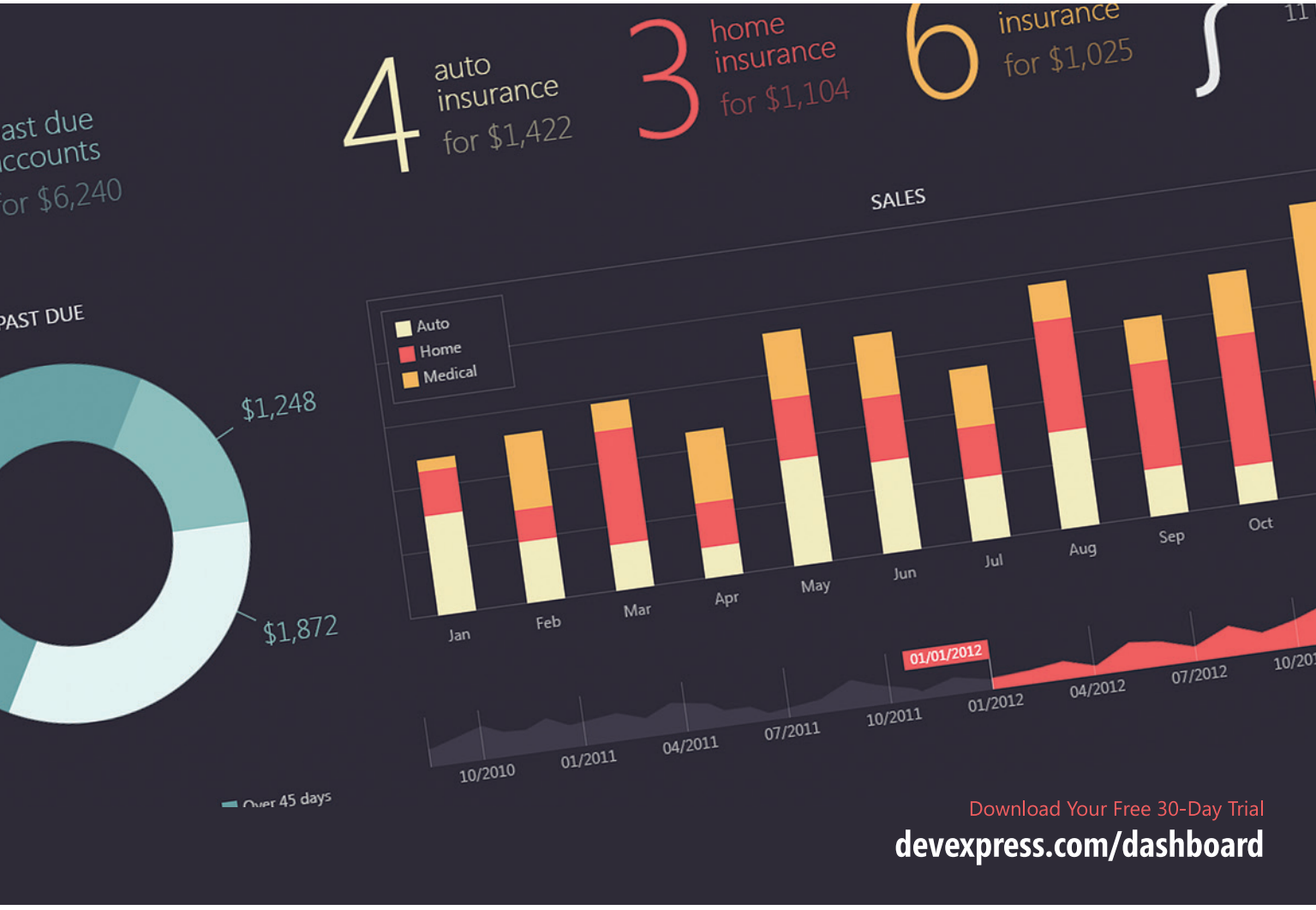
JANUARY 2017 VOL 32 NO 1

msdn magazine

C++

Your Next Great Dashboard Starts Here

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



Download Your Free 30-Day Trial

devexpress.com/dashboard

DevExpress

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

DevExpress

Your Next Great Desktop App Starts Here

From apps that replicate the look and feel of Microsoft Office® to high-powered data mining and decision support systems for your enterprise, DevExpress desktop controls 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

THE MICROSOFT JOURNAL FOR DEVELOPERS

JANUARY 2017 VOL 32 NO 1

msdn magazine

C++

C++ Language Projection for WinRT.....12

COLUMNS

Microsoft

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 application.

Follow the trend and switch to flow type layout reporting.

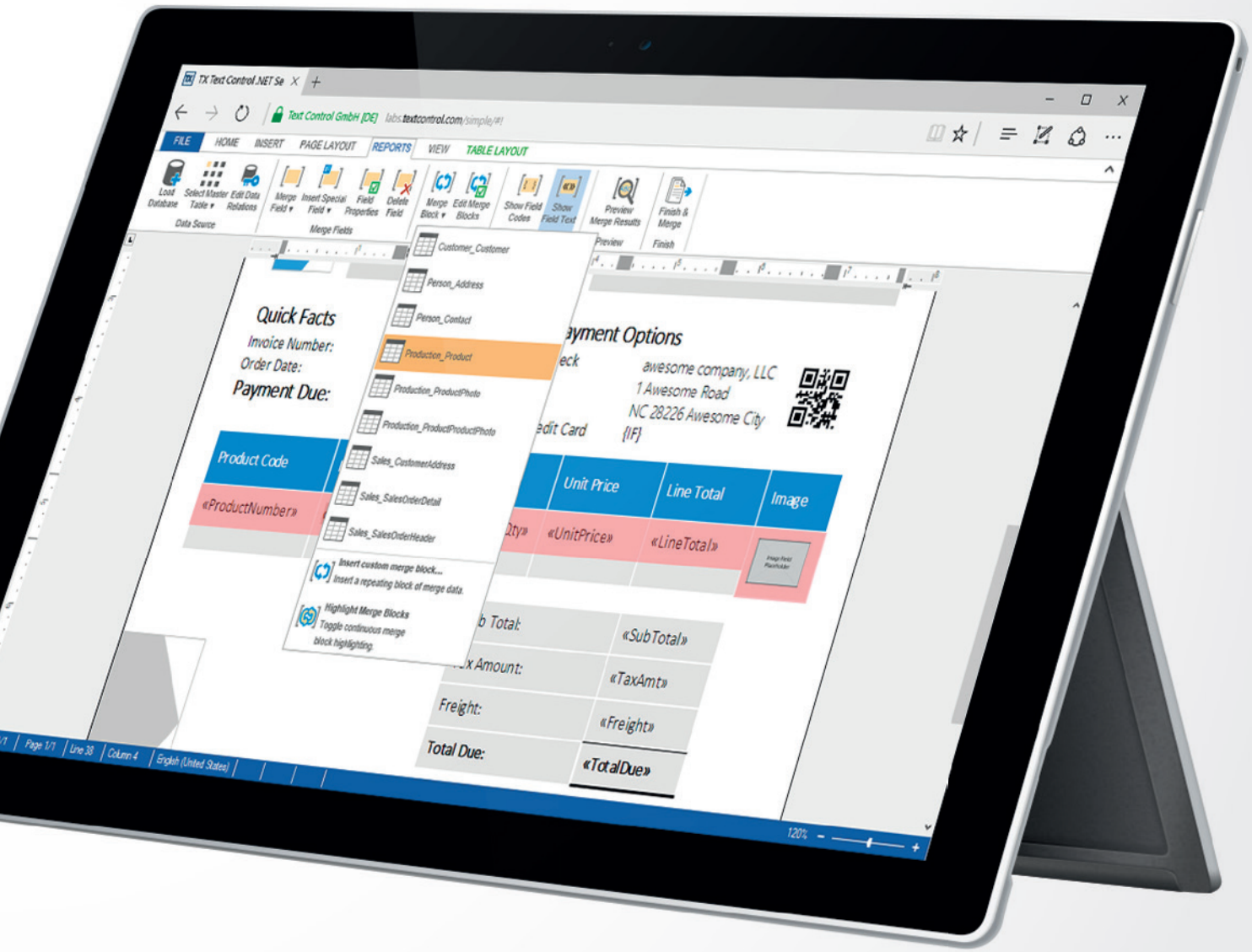
www.textcontrol.com www.reporting.cloud

Microsoft .NET WINDOWS FORMS | WPF | ASP.NET MVC | ASP.NET AJAX | CLOUD WEB API

Visual Studio Partner

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

REPORTING LIBRARIES FOR WINDOWS, WEB, MOBILE AND CLOUD APPLICATIONS



TEXT CONTROL

msdn magazine

JANUARY 2017 VOLUME 32 NUMBER 1

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

ENTERPRISE COMPUTING GROUP

President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

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

PRODUCTION STAFF
Print Production Coordinator **Lee Alexander**

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

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

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

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**

1105 MEDIA INC
YOUR GROWTH. OUR BUSINESS.

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: jlong@meritdirect.com; Web: www.meritdirect.com/1105

Reaching the Staff

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

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

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

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

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

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

WORLD-CLASS MOBILE RECOGNITION SDK

LEADTOOLS recognition engines enable developers to easily integrate fast and accurate OCR, Barcode, Driver's License, Passport, and Credit Card recognition functionality into their mobile apps.

Download fully functional sample apps built with the LEADTOOLS SDK.

Google play

Download on the App Store

Microsoft

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Outside in: How Kenny Kerr Brought C++/WinRT to Microsoft

C++ developers hoping to get in on the action with the Windows Runtime (WinRT) have faced a high barrier to entry. Either work with the Windows Runtime C++ Template Library (WRL) originally developed to bootstrap internal development of APIs by Microsoft, or turn to Microsoft's C++ Component Extensions (C++/CX) to streamline access to WRL capabilities, at the cost of having to learn a new dialect of C++.

Kenny Kerr, a longtime C++ columnist at *MSDN Magazine* and now an engineer on the Windows team at Microsoft, believed there had to be a better way. He started work early in 2014 on a Windows Runtime projection for standard C++ that would make the language a first-class citizen in the WinRT space.

"I had some previous experience projecting COM APIs into modern C++, so I decided to see whether I could apply those same techniques to the Windows Runtime," Kerr says. "I was invited to Redmond on two different occasions and it certainly seemed as if they were warming up to the idea. Finally, I was offered a job and joined the Windows team to complete the project with their help."

The result of that effort is C++/WinRT, a standard C++ language projection for WinRT that is implemented entirely in header files and enables developers to both consume and author Windows Runtime APIs using any standards compliant C++ compiler. This is exciting stuff, and Kerr explores it all in detail in his feature article this month, "Introducing C++/WinRT."

Kerr explains that there are really two parts to C++/WinRT—the base library and projection that can be downloaded from GitHub (URL), and the cppwinrt.exe compiler that bridges the gap between standard C++ and WinRT. While most of the effort to date has focused on the library and projection, Kerr says the cppwinrt compiler deserves attention. "Developers really need to get their hands on this tool as it solves a lot of problems, from generating projections for different platforms or components, to playing a key role in developing WinRT components entirely with C++/WinRT."

One of the biggest challenges Kerr faced in developing C++/WinRT was managing the trade-offs that WinRT makes to support projections for JavaScript and managed .NET languages out of the box. He singles out the complexity in how generic collections work across language projections.

"One of the biggest challenges I faced early on was coming up with an efficient way for standard C++ to handle WinRT's interface-versioning model. I mention this briefly in this month's article, but I don't touch on how those 'required' interfaces are actually aggregated together at compile time in C++," Kerr says. "It really pushed my understanding of C++ at that time and has since pushed the Visual C++ compiler to more efficiently handle such techniques at this scale."

Kenny Kerr, a longtime C++ columnist at *MSDN Magazine* and now an engineer on the Windows team at Microsoft, believed there had to be a better way.

Microsoft for years has been emphasizing openness and cross-platform support in its dev tools (see my Editor's Note column in the recent *MSDN Magazine* Connect(); special issue at msdn.com/magazine/mt790179). The C++/WinRT project is a great example of these values at work, as Microsoft welcomes the best efforts of the community in an effort to improve its platforms. Still, the question begs: How does a guy outside of Microsoft end up creating vital tooling for a mainline Microsoft programming language? Kerr says it comes down to fresh eyes.

"There are a lot of very experienced C++ developers at Microsoft who have spent decades-long careers working with C++ and COM," Kerr says. "I think it took someone who didn't realize that it was impossible to just try it anyway and show that it works."

© 2017 Microsoft Corporation. All rights reserved.

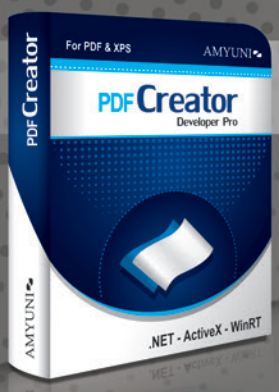
Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly 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.

Switch to Amyuni PDF

AMYUNI



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

NEW v5.5

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

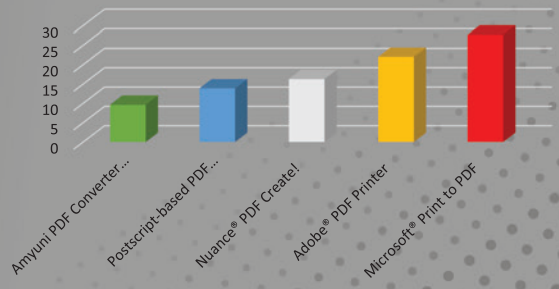
Complete Suite of Accurate PDF Components

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

High Performance PDF Printer for Desktops and Servers

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

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



Other Developer Components from Amyuni®

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

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

All development tools available at

www.amyuni.com

AMYUNI Technologies

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

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

You're Hired: 11 Things to Consider with Side Projects

Developers make crucial mistakes when listing technical "side projects" on their resumes. I see it all the time. Here are 11 questions to ask yourself before touting your side projects:

1. Why mention it? Most side projects are tiny hobby projects that simply aren't worth being listed on a resume. They'll get ignored. To stand out, position a side project so that it startles the interviewer into thinking, "Wow, this person really enjoys software engineering and is good at it."

2. Does it fulfill relevant experience? If you have little experience relevant to a job, a side project can be a powerful conversational pivot during interviews. Just be sure to list it under your work history if you can, rather than in a special Projects section. When you list something in Projects, you unintentionally discount its value by 75 percent to interviewers. You're telling them, "Don't pay attention to this section; it's just something little I did on the side."

If you have little experience relevant to a job, a side project can be a powerful conversational pivot during interviews.

3. Is it strong? An ideal project is as strong as your strongest "work" project, with similar characteristics: worked in a team, dealt with clients or customers, hit massive success metrics via downloads or financial gain, achieved press recognition, or overcame obstacles.

4. What's the story? No matter what side project I put on the resume, I'd craft four to five stories to handle almost any question about it. With every story, I'd make the side project the glorious, heroic solution that solved a crisis problem. I'd repurpose the stories when interviewers asked for my greatest strength, greatest weakness or obstacles overcome.

5. What was superlative about it? Figure out ways to describe your projects in terms of "bests." In what tiny way was your project the biggest, the most popular, the most downloaded, the most viewed, the fastest, or the most praised?

Almost any side project can be described superlatively yet truthfully in at least one way. Mastering this delicate, hyper-creative dance adds weight to your projects' value.

6. Is the right aspect emphasized? If I have to choose among many side projects, I pick one that shows me in the most outrageously positive light, be it in technical complexity, success achieved, scale, reach, results or relevance to the company for which I'm applying.

7. Did it help anyone? I recommend listing side projects under Work Experience when you solve at least one other person's problem with your project. This approach worked for my client Felicia, who used to list her side projects in a Projects section. After she moved them to her Work Experience section, and described them as "Client Engagements" (because she'd created all her side projects for at least one other person's benefit), she got an offer within a few weeks.

8. Is a Web site even necessary? Most recruiters barely take time to read your resume before an interview, much less find time to visit your Web sites. And if they do visit your Web site, they are unlikely to dig into your side projects; they'll just look for glaring red flags (like inappropriate content).

Candidates tend to over-share on their sites, in ways that can sabotage their candidacies. For most candidates, the risks of a personal Web site outweigh the rewards. If you have a personal site on your resume, make it 100 percent professional.

9. Where do you host side projects? It's not necessary to host projects online, but if you do, choose a site that gives you the most "street cred" for the role. For a UI developer, a gorgeous, professional portfolio site built on your own domain is the way to go. For other developers, GitHub is the best choice.

10. How does this project help the company? Sometimes, it's best to create a side project *specific* to the employer to which you're applying. For example, I may look into issues a company's customers are reporting, and create a project or mock-up that addresses some top concerns. I'd then share this with my interviewers, via my recruiter. This lets me set the agenda for the interview before I even have the interview. I can come off as an "internal" candidate even while coming from the outside.

11. Is this a cover-up? Side projects can even become an excuse for avoiding the discomfort of a job search. Many candidates busy themselves with side projects that do zero to advance their candidacy.

It's OK to list no side projects on a resume, especially if you've been working as a software engineer already. Some developers mistakenly think that one more side project (or the "right" side project) will produce more job offers. That's not the case, usually. That time is often better spent reaching out to the right hiring managers, preparing for interviews or sharpening the resume. ■

KRISHNAN RANGACHARI *is a career coach for software engineers. Visit RadicalShifts.com for his free courses and ByteshiftResumes.com for his resume makeovers.*

nsoftware

We didn't invent the Internet...

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

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

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

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

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

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

connectivity
powered by

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

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

JULIE LERMAN

EF Core 1.1: A Few of My Favorite Things

As I'm writing this column (in November 2016), Entity Framework (EF) Core 1.1 was just released. Between the 1.0 release and 1.1, a few significant things happened. In particular, the 1.0.1 patch fixed some critical bugs that were discovered just as 1.0 was being released. For a detailed list of those fixes, you can read the release notes at bit.ly/2fl5xNE. While most of these fixes relate to how LINQ queries are transformed to SQL, there are also fixes to the model builder, persistence via SaveChanges and migrations.

EF Core 1.1 brings additional bug fixes, many more improvements to how LINQ queries are interpreted, and big performance gains. But there are also new features—some that existed in EF6 but hadn't made it to EF Core 1.0, keeping many developers from even looking at EF Core—and others that are completely new. What I want to do in this article is present a quick list of these changes and then highlight some that are of great interest to me. I'll also provide links to the many valuable resources I use in my effort to keep up with EF Core as it evolves. To begin with, plenty of this information is covered in the docs (docs.microsoft.com/ef) and in the team blog posts (bit.ly/2ehZBUB).

In particular, you should be aware of Arthur Vickers' blog. Arthur is a senior dev who has been on the team since the inception of POCO support in EF and has been instrumental in the modernization of EF. He has a great series of posts on EF Core 1.1 at blog.oneunicorn.com.

Microsoft plans to bring many EF6 features to EF Core, but not all of them. And there are a number of new features planned that will come in future versions. Right now, EF Core 1.1 focuses on upgrades that would make EF Core more satisfactory to a greater number of developers. Here are the most important of these features.

EF6 Features Now in EF 1.1

The DbSet.Find method, which became available with the introduction of the DbContext API in EF 4.1, didn't make it onto the hit list for the first iteration of EF Core, upsetting a lot of developers. Find is more than a convenient way to efficiently query for an entity based on its key value. It first checks to see if that entity is already in memory and is being tracked by EF. If the entity can't be found in the DbContext cache, EF executes a FirstOrDefault query on the database to retrieve the entity. This used to be a SingleOrDefault query in EF6 and earlier. Find's design to avoid an unnecessary trip to the database is also a performance benefit. If you're like me and want to see the plumbing, you can find the code in the EntityFinder class on GitHub (bit.ly/2e9VOUu).

The other big difference is that Find is now in an EntityFinder service; it's not just a method implemented directly in an internal DbSet class as it was in EF6. EF Core is made up of hundreds of services to encapsulate particular tasks. In his demos in the Channel 9 video on EF Core 1.1 (bit.ly/2fHOCsN), Rowan Miller shows how to use services directly, as well as how to replace them.

> EF has always provided three ways of loading related data.

Also becoming part of EF Core with the 1.1 update is the EF6 feature referred to as *connection resiliency*, which provides support for easily handling transient connection problems that can happen when working against a remote database such as Azure SQL Database. The EnableRetryOnFailure extension method of the SQL Database provider can be set in DbContext.OnConfiguring or the AddDbcontext call in Startup.cs if you're using ASP.NET Core dependency injection. EnableRetryOnFailure calls SqlServerRetryingExecutionStrategy, which inherits from the ExecutionStrategy type. You can create and set your own ExcecutionStrategy and other providers can predefine their own ExecutionStrategy configurations, as well. If you're not familiar with this feature, it's similar to the EF6 DbExecutionStrategy, which I covered in depth in my Pluralsight course, "EF6 Ninja Edition" (bit.ly/PS_EF6).

EF has always provided three ways of loading related data. One, eager loading, is enabled by the DbSet.Include method to retrieve graphs of data in a single query. Two others, in contrast, load related data after the main objects are already in memory and while the DbContext that's tracking them is still in scope. Lazy loading automatically pulls in related data on-demand, while with explicit loading you explicitly tell EF to load the related data. Include was the first of these to be implemented in EF Core and became available with EF Core 1.0. It even has some nice improvements over versions from EF6 and earlier. Explicit loading with the Load method has now been added into EF Core 1.1. Lazy loading is not yet supported but will be at some point.

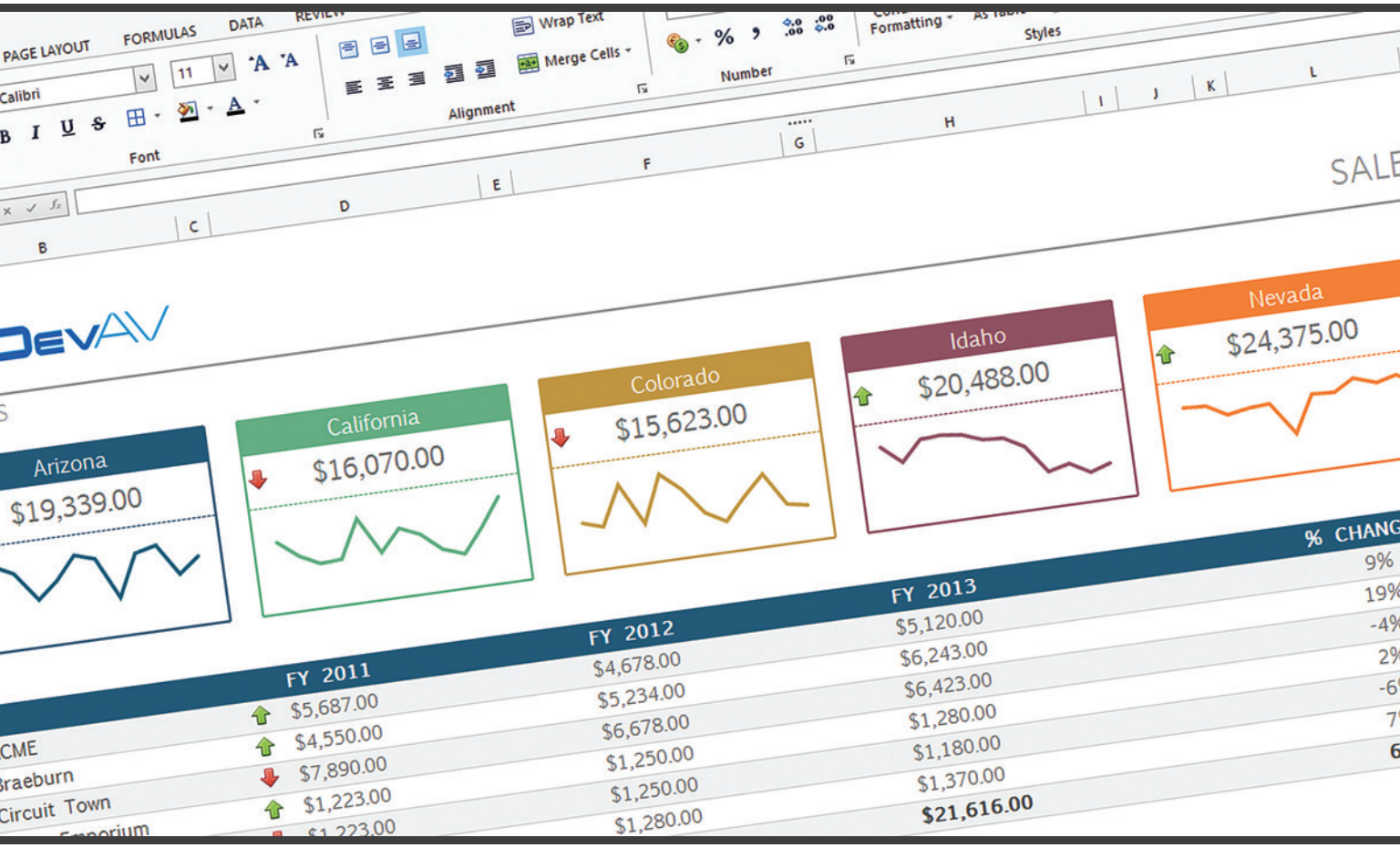
DbContext's change tracker APIs let you access change tracker information directly, for example getting or setting the state of an entity with the DbContext.Entry(someObject).State method. EF6

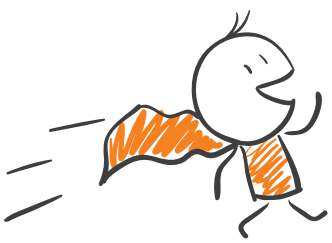
Code download available at msdn.com/magazine/0117magcode.

DevExpress®

Office® Inspired Apps

Get started today and create high-performance, high-impact .NET solutions that fully replicate the look, feel and user-experience of Microsoft Office.®





Your Next Great Business App Starts Here

Explore our complete range of Office-inspired controls for all major .NET platforms.
devexpress.com/office

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

brought additional control with new methods like GetDatabaseValues, CurrentValues and OriginalValues. These methods are now available in EF Core 1.1.

New Capabilities in EF 1.1

EF Core is filled with features that never existed in earlier versions. Here's a short list of examples: batching during SaveChanges, unique foreign keys, the wonderful InMemory provider for testing, smarter LINQ query processing, and smarter and simpler fluent mappings.

EF 1.1 brings some additional new features and there's one in particular that, as someone who's a big fan of Domain-Driven Design (DDD), I'm quite fond of—support for encapsulated collections.

Mapped Fields and Encapsulated Collections EF Code First has only ever supported mapping to properties that have both a getter and a setter, even if the setter is private. And for collection navigations, the property was required to be an ICollection. If you want to constrain how the values of properties are populated, the ability to encapsulate the property is crucial—that way you can force anyone using your class to use a public method to ensure that the business rules around that property are honored. EF6 and earlier allow you to encapsulate scalar properties by making the setter private. But there's no way to truly encapsulate collections and prevent anyone from modifying the collection directly.

> With EF 1.1, you gain the ability to map directly to fields, as well as to map to IEnumerable properties.

With EF 1.1, you gain the ability to map directly to fields, as well as to map to IEnumerable properties. The new ability to map to fields, not just properties, allows you to use a more direct approach than hiding the setter and it also supports some additional ways of encapsulating scalar values. Here's a property, DueDate, that has a getter but no setter, along with the field, _dueDate, which is tied to the property:

```
private DateTime _dueDate;
public DateTime DueDate {
  get { return _dueDate;
  }
}
```

The only way to set the DueDate is by calling the CalculateDueDate method, which modifies the field:

```
private void CalculateDueDate() {
  _dueDate=Start.AddDays(DaysLoaned);
}
```

EF Core 1.1 requires an explicit mapping in the DbContext to inform EF that it can use the _dueDate field to map to the database, for example when returning the results of a query. You must use the Property (and optionally HasField) API methods to specify that the _dueDate field is a stand-in for the DueDate property. In this case, because the field name, _dueDate, follows an EF convention, I don't have to use the HasField method, but I've added it in so you can see it:

```
protected override void OnModelCreating(ModelBuilder modelBuilder) {
  modelBuilder.Entity<BookLoan>().Property(bl => bl.DueDate)
  .HasField("_dueDate");
}
```

While you could use private setters before the field mapping was available, there was no way to encapsulate collections, to prevent anyone from directly manipulating a collection via its Add or Remove methods. It isn't a matter of hiding the setter; what's needed is to hide the collection methods. A common approach in DDD is to make the property an IEnumerable. However, with EF6 and earlier, you can only map to a flavor of an ICollection; but IEnumerable isn't an ICollection.

At first I looked to see if you could use the field-mapping capability, but with EF 1.1 that isn't possible as it only supports mapping to scalar properties. This is targeted to change in the next release of EF Core, which will allow mapping to navigation properties. But when I pointed this out one day on Twitter, Arthur Vickers let me know that in fact you can map to IEnumerables—something I'd somehow missed about EF Core. So I'm now able to encapsulate and protect a collection completely, forcing users of my API to go through a method to modify the collection. Here's an example where I can add new BookLoan instances to a book every time it's loaned, ensuring that the period of the loan is included:

```
private List<BookLoan> _bookLoans;
public IEnumerable<BookLoan> BookLoans {
  get { return _bookLoans; }
}
public void BookWasLoaned(int daysLoaned){
  _bookLoans.Add(new BookLoan(DateTime.Today, 14));
}
```

This is one pattern for achieving encapsulation by leveraging the IEnumerable mapping. But I recommend you read Arthur's blog post (bit.ly/2fVzIfN) for more details, including how to do this without a backing field (a la my _bookLoans field), plans for enhancements and some gotchas to watch out for.

Support for Database-Specific Features EF Core is designed to allow providers to more easily support specific features of the data store. One example is that the SQL Server provider for EF Core supports SQL Server Memory-Optimized Tables for applications where you have incredibly high throughput. To specify that an entity maps to this special type of table, the SQL Server provider has an extension method you can use when configuring your model with the Fluent API:

```
modelBuilder.Entity<Book>().ForSqlServerIsMemoryOptimized();
```

Not only will this affect how code first generates table creation script, but it will also affect how EF generates commands to push data into the database. You can see an interesting demo of this in the previously mentioned Channel 9 video where Rowan Miller demos EF 1.1 features.

Shay Rojansky, who built the PostgreSQL provider for EF Core, wrote an article about how EF Core allowed him to support special features of PosgreSQL such as array types. Read it at bit.ly/2focZKQ.

Easier Access to Services

As I highlighted earlier with the EntityFinder service, EF Core is made up of hundreds of services. In the Channel 9 video, Rowan demonstrates how to access and use these services directly in your code. In addition, you can override a service with your own customi-

zation. EF 1.1 makes it easier to do that with a simple replace method that can be used in OnConfiguring. You can replace a service with another that either inherits from that service or implements the same interface. There's no particular list of services—these are just classes throughout the various APIs of EntityFrameworkCore. A simple-to-grasp example is to create a class that inherits from a database type mapper, such as SqlLiteTypeMapper from the Sqlite provider, and add in a new type-mapping rule. Be sure that rule is one the database is able to cast. (Some cleverer type conversions will be coming in a future version of EF Core.) Then, in OnConfiguring, set up the replacement rule:

```
optionsBuilder.ReplaceService<SqliteTypeMapper,MySqliteTypeMapper>();
```

Why did I not replace the EntityFinder service? First, because I like the way it works. But second, because it's a generic class, which makes creating a new version of it more complicated and I chose to set that chore aside for the time being. While ReplaceService was created to make it easier to replace internal services, you do need to keep in mind that EF Core internals could change and, therefore, your replacement might be problematic down the road. You can identify these easily enough because they're in namespaces that end with the word Internal. From Microsoft: "We generally avoid breaking changes in non .Internal namespaces in minor and patch releases."

Worth highlighting (because it caused a bit of a stir at the time) is a fix to a performance problem related to asynchronous queries that used the Include method, which was discovered just as 1.0 was about to be released. It was addressed quickly (see bit.ly/2faltD1 if you're interested in the details) with a reported 70 percent increase in performance and is also part of the 1.1 release.

Wrapping Up

EF Core has a lot of sophisticated features. It's incredibly important to be aware of what's in, what's not in, what's coming and what's never going to be part of EF Core. The EF Core documentation about its relationship to EF6 (bit.ly/2fxbVWj) is a useful resource, as well. The point is, you decide when EF Core is right for you and for your apps.

For most of the work I do, EF Core 1.1 has the APIs and features I need. As a DDD practitioner, the ability to encapsulate collections is a winner for me, though I'm still waiting for the complex type mappings to be included so I can also use value objects in my models. This feature is slated for the next release of EF Core. I was also excited to recently discover that EF Core has fulfilled a wish of mine—to define a read-only (non-tracking) DbContext. I had completely missed that this was added early on and was part of the 1.0 release of EF Core. You can read more about that in my blog post at bit.ly/2f75l8m. ■

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

Thanks to the following Microsoft technical expert for reviewing this article: *Rowan Miller*

dtSearch®

Instantly Search Terabytes of Text

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

Highlights hits in all data types; 25+ search options

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

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

The Smart Choice for Text Retrieval® since 1991

dtSearch.com 1-800-IT-FINDS

Introducing C++/WinRT

Kenny Kerr

The Windows Runtime (WinRT) is the technology behind the modern Windows API, and the core of the Universal Windows Platform (UWP). The same API can be used across all Windows devices, whether it's a PC, tablet, HoloLens, phone, Xbox or anything else that runs Windows.

WinRT is based on the Component Object Model (COM), but its COM APIs aren't designed to be used directly, as you might use classic COM APIs such as those in DirectX. Rather, it's designed to be used through what are called "language projections." A language projection encapsulates the unpleasant details of working with the COM APIs and provides a more natural programming experience for a particular programming language. For example, the WinRT APIs can be used from both JavaScript and .NET quite naturally, without having to worry too much about the COM underpinnings.

Until recently, there hasn't been a good language projection for use by C++ developers: You had to choose between the clunky C++/CX language extensions or the verbose, tedious and complex WinRT C++ Template Library (WRL).

This is where C++/WinRT comes in. C++/WinRT is a standard C++ language projection for WinRT implemented entirely in header files—the best kind of C++ library. It's designed to support the authoring and consumption of WinRT APIs using any standards-compliant C++ compiler. C++/WinRT finally provides C++ developers with first-class access to the modern Windows API.

Windows Metadata Files to the Rescue

C++/WinRT is based on the Modern C++ for the Windows Runtime project (moderncpp.com), a project I started prior to joining Microsoft. It was in turn based on another project I created in an attempt to modernize DirectX programming (dx.codeplex.com). When WinRT came along, it solved the No. 1 problem of modernizing COM APIs by providing a canonical way of describing the API surface, via what are called Windows Metadata (.winmd) files. Given a set of Windows Metadata files, the C++/WinRT compiler (cppwinrt.exe) can generate a standard C++ library that fully describes or projects the Windows API, or any other WinRT components, so that developers can both consume and produce implementations of that API. The latter is significant because it means that C++/WinRT is not only for calling or using WinRT APIs, but it's also ideally suited to implementing WinRT components. Teams within Microsoft are already starting to use C++/WinRT to build WinRT components for the OS itself.

C++/WinRT had its first public release on GitHub in October 2016, so you can try it today. Perhaps the simplest option is to clone the git repository, although you can also download it as a .zip file. Simply issue the following command:

```
git clone https://github.com/Microsoft/cppwinrt.git

Cloning into 'cppwinrt'...
```

You should now have a folder called "cppwinrt" that contains your own local copy of the repository. This folder contains the C++/WinRT library, as well as some documentation and a getting started guide. The library itself is contained within a folder reflecting the version of the Windows SDK against which it was built. As of this writing, the latest version is 10.0.14393.0, which is the build of the Windows SDK that supports development for the Windows 10 Anniversary Update (RS1). Your local folder might look something like this:

```
dir /b cppwinrt

10.0.14393.0
Docs
Getting Started.md
license.txt
media
README.md
```

The versioned folder itself contains two folders:

```
dir /b cppwinrt\10.0.14393.0

Samples
winrt
```

The winrt folder is all you need. If you simply want to copy that folder into your own project or source control system, that's perfectly fine. It contains all the required headers, so you can quickly start writing code. What does that code look like? Well, let's start with a console app that you can simply compile using the Visual C++ tools command prompt. No need for Visual Studio or those complicated

This article discusses:

- A new Windows Runtime language projection for standard C++
- How C++ can be used to modernize COM programming
- The effectiveness of coroutines in simplifying concurrency

Technologies discussed:

Windows Runtime, Standard C++, Windows Metadata, Coroutines

.msbuild or .appx files that are staples of your typical UWP app. Copy the code from **Figure 1** into a source file and call it Feed.cpp.

I'll talk in a moment about what the code means. For now, you can make sure it builds using the following Visual C++ compiler options:

```
cl Feed.cpp /I cppwinrt\10.0.14393.0 /EHsc /std:c++latest

Microsoft (R) C/C++ Optimizing Compiler...
```

The /std:c++latest compiler flag is the clue that you'll need Visual C++ 2015 Update 3 or later to use C++/WinRT. Notice, however, that there's no Microsoft-specific extension being requested. In fact, you might also want to include the /permissive- option that further constrains your code to be more standards-compliant. If all goes well, the compiler will call the linker and together produce an executable called Feed.exe that prints out the titles of my recent blog posts:

```
Feed.exe

C++/WinRT: Consumption and Production
C++/WinRT: Getting Started
Getting Started with Modules in C++
...
```

On with the Show

Congratulations! You've built your first app with C++/WinRT. So, what's going on here? Take another look at **Figure 1**. The first line tells the linker where to find a handful of WinRT functions used by any language projection. These aren't specific to C++/WinRT. They're merely the Windows APIs that let an app or component initialize the apartment context for a thread, manipulate WinRT strings, activate factory objects, propagate error information and so on.

Next up is a set of #include directives to include the types from the namespaces that the program intends to use. Originally, C++/WinRT defined everything within a single header file, but, given the sheer size of the Windows API, this proved to be detrimental to build throughput. In the future, we might switch to using C++ modules as I described in my April 2016 *MSDN Magazine* article, "Microsoft Pushes C++ into the Future" (msdn.com/magazine/mt694085). When that day comes, having to first include these headers might no longer be necessary because the module format is so much more efficient and early trials have indicated that the build throughput issues might largely disappear.

The using namespace directives are optional, but given that the Windows API uses namespaces quite heavily, they do come in rather handy. All the types along with their enclosing namespaces are housed within the C++/WinRT root namespace called winrt. This is somewhat unfortunate but required for interoperability with existing code: Both C++/CX and the Windows SDK declare types in a root namespace named "Windows." Having a separate namespace lets developers slowly migrate away from C++/CX while preserving their existing investment in these older technologies.

The app in **Figure 1** simply creates a Uri object to indicate the RSS feed to download. The Uri object is defined in the Windows.Foundation namespace. This Uri object is then passed to a method of the SyndicationClient object to retrieve the feed. SyndicationClient is defined in the Windows.Web.Syndication namespace. As you'll see in a moment, WinRT relies heavily on async, so the app needs to wait for the result of the RetrieveFeedAsync method and that's the job of the trailing get method. With the SyndicationFeed object in hand, the feed's items may be enumerated via a collection returned by the Items method. Each resulting SyndicationItem may then be unpacked to retrieve the text from the post's title. The result is of type hstring, which encapsulates the WinRT HSTRING type, but provides an interface similar to that of std::wstring, to provide a familiar experience to the C++ developer.

As you can see, WinRT aims to present a classy type system over the tried-and-tested COM plumbing at the heart of so much of the Windows OS. C++/WinRT faithfully honors that ideal. There's no need to call arcane functions like CoCreateInstanceEx or deal with COM pointers or HRESULT error codes. Of course, the moral equivalent of these COM-style primitives still exists behind the scenes. For example, the initialize function called at the start of the main function in **Figure 1** internally calls the RoInitialize function, which is the WinRT equivalent of the traditional CoInitializeEx function from classic COM. The C++/WinRT library also takes care of converting between HRESULTs and exceptions to provide a natural and productive programming model for the developer. This is done in a manner so as to avoid code bloat and improve inlineability.

I alluded to the fact that WinRT has a deep architectural investment in async and you saw this already in the example in **Figure 1**. The RetrieveFeedAsync function might naturally take some time to complete and the API designers didn't want the method call to block, so rather than returning a SyndicationFeed directly it instead returns an IAsyncOperationWithProgress type that represents the operation that might eventually result in a SyndicationFeed when those results are finally available.

Concurrency

WinRT provides four types to represent different kinds of async objects and C++/WinRT provides a few ways to both create and consume such objects. **Figure 1** shows how you can block the calling thread until the results are available, but C++/WinRT also integrates C++ coroutines very deeply into the programming model to provide a natural way to cooperatively wait for the result without holding up OS threads from other useful work. Instead of using the get method, you can write a co_await statement to await the result of the operation as follows:

```
SyndicationFeed feed = co_await client.RetrieveFeedAsync(uri);
```

Figure 1 **Your First C++/WinRT App**

```
#pragma comment(lib, "windowsapp")
#include "winrt/Windows.Foundation.h"
#include "winrt/Windows.Web.Syndication.h"

using namespace winrt;
using namespace Windows::Foundation;
using namespace Windows::Web::Syndication;

int main()
{
  initialize();

  Uri uri(L"http://kennykerr.ca/feed");
  SyndicationClient client;
  SyndicationFeed feed = client.RetrieveFeedAsync(uri).get();

  for (SyndicationItem item : feed.Items())
  {
    hstring title = item.Title().Text();
    printf("%ls\n", title.c_str());
  }
}
```

You can also produce your own WinRT async object simply by writing a coroutine as illustrated in **Figure 2**. The resulting IAsyncAction, another WinRT async type found in the Windows.Foundation namespace, may then be aggregated into other coroutines, a caller might decide to use the get method to block and wait for the result, or it can even be passed to another programming language that supports WinRT.

As I've mentioned, C++/WinRT is not just about *calling* WinRT APIs. It's just as easy to implement WinRT interfaces and produce implementations that others might call. A good example of this is in the way the WinRT application model is structured. A minimal app might look something like this:

```
using namespace Windows::ApplicationModel::Core;

int __stdcall wWinMain(HINSTANCE, HINSTANCE, PWSTR, int)
{
  IFrameworkViewSource source = ...

  CoreApplication::Run(source);
}
```

That's just the traditional WinMain entry point function reminding you that even though you can now write in standard C++, you're still in the business of writing apps that shine on specific platforms. The CoreApplication's static Run method expects an IFrameworkViewSource object to create the app's first view. IFrameworkViewSource is just an interface with a single method and looks, at least conceptually, like this:

```
struct IFrameworkViewSource : IInspectable
{
  IFrameworkView CreateView();
};
```

So, given an IFrameworkViewSource, you might call it as follows:

```
IFrameworkViewSource source = ...
IFrameworkView view = source.CreateView();
```

Of course, it's not the app developer who calls it but the operating system. And you, the app developer, are expected to implement this interface, along with the IFrameworkView interface that CreateView returns. C++/WinRT makes it very simple to do so. Again, there's no need for COM-style programming. You don't need to resort to scary WRL or ATL templates and macros. You can simply use the C++/WinRT library to implement interfaces as follows:

```
struct Source : implements<Source, IFrameworkViewSource>
{
  IFrameworkView CreateView()
  {
    return ...
  }
};
```

The implements class template is loosely based on the variadic template approach to implementing WinRT interfaces. I described this in my 2014 Special Connect(); article, "Visual C++ 2015 Brings Modern C++ to the Windows API" (msdn.com/magazine/dn879346).

The first type parameter is the name of the derived class that intends to implement the interfaces listed as subsequent type parameters. How do you implement the IFrameworkView interface? Well, it's another interface that's only slightly more complicated. It looks something like this:

```
struct IFrameworkView : IInspectable
{
  void Initialize(CoreApplicationView const & applicationView);
  void SetWindow(Windows::UI::Core::CoreWindow const & window);
  void Load(hstring_ref entryPoint);
  void Run();
  void Uninitialize();
};
```

Given an instance of IFrameworkView, you can freely call those methods just as they're described here, but again you're talking about an interface that the app is expected to implement and the OS will call. Again, you can simply use C++/WinRT to implement this interface as illustrated in **Figure 3**. This represents a minimal app that will get a window up and running on the desktop even while it may not do anything exciting. I'm showing you all of this not to illustrate how to write the next great app, but to show how you can use natural C++ code to both consume and produce WinRT types.

You can then update the IFrameworkViewSource implementation to make and return this implementation:

```
struct Source : implements<Source, IFrameworkViewSource>
{
  IFrameworkView CreateView()
  {
    return make<View>();
  }
};
```

You can likewise update your WinMain function to make use of your IFrameworkViewSource implementation as follows:

Figure 2 **Coroutines with C++/WinRT**

```
IAsyncAction PrintFeedAsync()
{
  Uri uri(L"http://kennykerr.ca/feed");
  SyndicationClient client;
  SyndicationFeed feed = co_await client.RetrieveFeedAsync(uri);

  for (SyndicationItem item : feed.Items())
  {
    hstring title = item.Title().Text();
    printf("%ls\n", title.c_str());
  }
}

int main()
{
    initialize();
    PrintFeedAsync().get();
}
```

Figure 3 **Implementing IFrameworkView**

```
struct View : implements<View, IFrameworkView>
{
  void Initialize(CoreApplicationView const & view)
  {
  }

  void Load(hstring_ref entryPoint)
  {
  }

  void Uninitialize()
  {
  }

  void Run()
  {
    CoreWindow window = CoreWindow::GetForCurrentThread();
    window.Activate();

    CoreDispatcher dispatcher = window.Dispatcher();
    dispatcher.ProcessEvents(CoreProcessEventsOption::ProcessUntilQuit);
  }

  void SetWindow(CoreWindow const & window)
  {
    // Prepare app visuals here
  }
};
```

BEST SELLER

DevExpress DXperience 16.1 | from $1,439.99

DevExpress

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

BEST SELLER

ActiveReports 11 | from $1,567.02

GrapeCity

Award-winning .NET reporting platform for HTML5, WPF, WinForms, ASP.NET & Windows Azure.

- Visual Studio-integrated report designer
- Extensible optional report server with built-in scalability
- Responsive HTML5 Report Portal
- Royalty-free redistribution
- Source data from JSON files, Web services and REST API using the built in JSON data provider

BEST SELLER

Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

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

BEST SELLER

Aspose.Total for .NET | from $2,939.02

ASPOSE
File Format APIs

Every Aspose .NET component in one package.

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

© 1996-2017 ComponentSource. All Rights Reserved. All prices correct at the time of press. Online prices may vary from those shown due to daily fluctuations & online discounts.

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

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

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

Asia / Pacific Headquarters
ComponentSource
7F Kojimachi Ichihara Bldg
1-1-8 Hirakawa-cho
Chiyoda-ku
Tokyo, 102-0093
Japan

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

GSA Schedule Contract GS-35F-0188R

```
int __stdcall wWinMain(HINSTANCE, HINSTANCE, PWSTR, int)
{
  CoreApplication::Run(make<Source>());
}
```

Given that the implements class template is designed as a variadic template, it can be used to implement multiple interfaces quite easily. You might decide to implement both IFrameworkViewSource and IFrameworkView inside one class, as shown in **Figure 4**.

If you've spent any time implementing COM objects with libraries such as ATL or WRL, this should be a welcome improvement. And it's not only more convenient, productive and enjoyable to use, but it will in fact give you the smallest binaries and the best possible performance of any WinRT language projection. It will even outperform handwritten code using the ABI interfaces directly. That's because the abstractions are designed to take advantage of modern C++ idioms that the Visual C++ compiler is designed to optimize. This includes magic statics, empty base classes, strlen elision, as well as many newer optimizations in the latest version of Visual C++ targeted specifically at improving the performance of C++/WinRT.

As an example, WinRT introduces the concept of required interfaces. A given runtime class or interface might be expected to additionally implement some other set of interfaces. This lets WinRT, which is based on COM interfaces that cannot ever change, support versioning. For example, the Uri class I illustrated earlier requires the IStringable interface with its single ToString method. A typical app developer isn't aware that the methods of the IStringable interface are in fact provided by a distinct COM interface and vtable and can simply make use of it as follows:

```
Uri uri(L"http://kennykerr.ca/feed");
hstring value = uri.ToString();
```

Behind the scenes, C++/WinRT must first query the Uri object for the IStringable interface using the IUnknown QueryInterface method. This is all fine and well until you happen to call such a required interface method in a loop:

```
Uri uri(L"http://kennykerr.ca/feed");

for (unsigned i = 0; i != 10'000'000; ++i)
{
  uri.ToString();
}
```

What you don't see is that this code results in something like this happening:

```
Uri uri(L"http://kennykerr.ca/feed");

for (unsigned i = 0; i != 10'000'000; ++i)
{
  uri.as<IStringable>().ToString();
}
```

C++/WinRT injects the necessary call to the as method, which in turn calls QueryInterface. Now, from the perspective of COM, QueryInterface is a pure call. If it succeeds once, it must always success for a given object. Fortunately, the Visual C++ compiler is now optimized to detect this pattern and in collaboration with C++/WinRT it will hoist this call out of the loop so that the code ends up looking like this:

```
Uri uri(L"http://host/path");
IStringable temp = uri.as<IStringable>();

for (unsigned i = 0; i != 10'000'000; ++i)
{
  temp.ToString();
}
```

Figure 4 **Implementing Multiple Interfaces with C++/WinRT**

```
struct App : implements<App, IFrameworkViewSource, IFrameworkView>
{
  // IFrameworkViewSource method...
  IFrameworkView CreateView()
  {
    return *this;
  }

  // IFrameworkView methods...
  void Initialize(CoreApplicationView const & view);
  void Load(hstring_ref entryPoint);
  void Uninitialize();
  void Run();
  void SetWindow(CoreWindow const & window);
};
```

This ends up being quite an important optimization as the number of required interfaces in the Windows API is quite significant, especially in areas such as XAML development. This is just one example of some of the incredible optimizations that are available to you as you adopt C++/WinRT for your app and component development. There are further optimizations that amortize the cost of accessing activation factories, resulting in significant improvements to the performance of instance activation (constructors) and static method calls that give you as much as 40x performance improvement over C++/CX.

Standard C++ presents some unique challenges to anyone attempting to produce a WinRT language projection, which is partly why the Visual C++ team at Microsoft originally came up with a non-standard solution. The work is not yet done, but we continue to work very hard on C++/WinRT with the goal of providing a first-class and no-compromise language projection for the systems programmer, the app developer—really any developer interested in writing beautiful and fast code for Windows.

Wrapping Up

James McNellis and I gave two talks at CppCon in 2016 where we officially unveiled C++/WinRT. You can find videos of those two talks here:

- "Embracing Standard C++ for the Windows Runtime" (bit.ly/2ePwbyz)
- "Putting Coroutines to Work with the Windows Runtime" (bit.ly/2fMLZqy)

These are rather technical "under the hood" presentations. For a high-level introduction, you can hear my CppCast interview at bit.ly/2fwF6bx.

Finally, you can download and give C++/WinRT a try today: github.com/microsoft/cppwinrt.

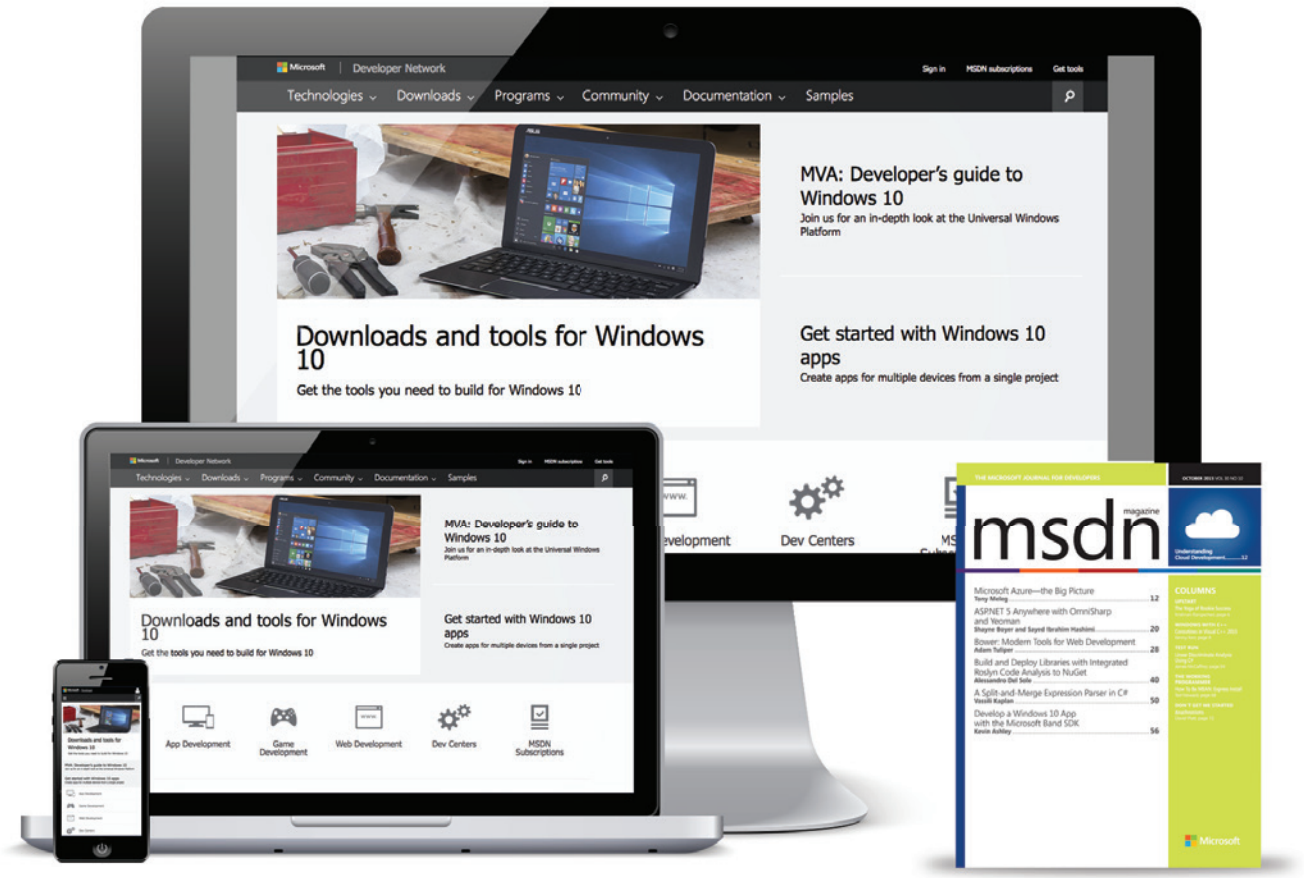
A lot of work remains to be done and there might yet be some changes to come in following the evolution of standard C++ and looking for additional ways to improve and simplify the programming model. Let me know what you think. I'd love your feedback.■

Kenny Kerr *is a C++ systems programmer, creator of C++/WinRT, an author for Pluralsight, and an engineer on the Windows team at Microsoft. He blogs at kennykerr.ca and you can follow him on Twitter: @kennykerr.*

Thanks to the following Microsoft technical expert for reviewing this article: *James McNellis*

msdn magazine

Where you need us most.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

Enable Natural Language Interaction with LUIS

Ashish Sahu

The technological landscape has changed quite radically in recent years. Computing capabilities moved from PCs to smartphones and wearables, while adding even more power to these devices. This advancement in technology has also changed the way we interact with our devices. Physical keyboards have been replaced with software implementations and the input methods changed from stylus to a mere tap of the fingers. It was only a matter of time before we started looking for even more effortless ways to interact with our computing devices.

Speech is how we interact with each other and now we're at the verge of using speech to also interact with all our smart devices. The recent launch of Bot Framework and Cognitive Services at the Build 2016 conference is a step toward that vision. Among these amazing cognitive services, Language Understanding Intelligence Service (LUIS) provides you with the capabilities to understand the natural language queries and return actionable information that you can wire up in your code to improve the UX.

In this article, I'll explore the capabilities of LUIS and look at different ways you can use it in your apps and services.

This article discusses:

- Conversational intelligence is poised to power the next wave of computing where users just need to ask for services and tasks
- LUIS provides the much talked about and needed technology block to enable conversational intelligence to your apps and services
- The SDKs and the RESTful API interface provide easier integration with apps regardless of programming language being used

Technologies discussed:

Microsoft Cognitive Services, Universal Windows Platform Apps, Bot Framework, Machine Learning

Code download available at:

bit.ly/2eEmPsy

How Does LUIS Work?

LUIS is built on the interactive machine learning and language understanding research from Microsoft Research. The book "Machine Learning" (McGraw Hill, 1997) by Tom Mitchell defines machine learning as:

> *"A computer program is said to learn to perform a task T from experience E, if its performance at task T, as measured by a performance metric P, improves with experience E over time."*

Like any other machine learning paradigm, LUIS is an iteration of this concept. It uses a language model and set of training examples to parse spoken language and return only the interesting parts that you, as developers, can use to delight your users.

With LUIS, apart from using your own purpose-specific language model, you can also leverage the same pre-existing and pre-built language models used by Bing and Cortana.

LUIS has a very specific use case—you can leverage LUIS anywhere you have a need to let users interact with your apps using speech. Most digital assistants, voice-enabled apps/devices, and bots fall into this category, but you're free to use your imagination.

Where Can I Use LUIS?

Using LUIS with your apps and services requires initial setup. The homework you need to complete consists of understanding the scenario and anticipating the interaction that'll take place between the apps and the users. Understanding and anticipating the interaction will help you build the language model to use with LUIS and to come up with the basic natural utterance to train LUIS to parse them.

Language Models

Language models are specific to your LUIS applications. They form the basis of understanding what the users mean when they talk to your apps and services. There are two core parts of a language model–"intents" and "entities." LUIS application uses the intents and entities to process the natural language queries and derive the intention and the topics of interest to the users with help from the training examples, also called "utterances." LUIS applications always contain a default intent called "None." This intent gets mapped to all the utterances that couldn't be mapped to any other intents. In the context of an app, intents are actions that the users intend to perform while the entities get filtered to the topics that your apps and services are designed to handle.

> Based on the examples and active learning, LUIS starts detecting the intents in the queries posted to it.

An example to understand this would be to imagine a shopping app and the following model:

```
"intents": [
  {
    "name": "ShowItems"
  }
  {
    "name": "BuyItems"
  }
],
"entities": [
  {
    "name": "Item"
  }
]
```

The majority of time spent in a shopping app might be with sale items and when someone says, "Show me red scarves," the model will map this utterance to ShowItems as the intent and red scarves to the entity item. At the same time, you can map an utterance to the BuyItems intent and thus initiate the checkout process when someone says, "I would like to pay now."

Intents

LUIS intents also support action binding, which lets you bind parameters to your intents. Actionable intents fire only when these parameters are provided, as well. In particular, when you use action binding with bots, LUIS can query the parameters from the users interactively.

Based on the examples and active learning, LUIS starts detecting the intents in the queries posted to it. However, because the language queries are tricky for the computer applications, LUIS also scores them between 0 and 1 to denote its confidence; higher scope denotes higher confidence.

Entities

LUIS entities, as explained here, are the topics of interest to your users. If you're building a news app, entities will map to the news topics, and in case of weather apps, they map to locations in their very basic iterations.

LUIS active learning also starts showing up when you add a new utterance and you can see the appropriate entities color-coded to show the mappings visually.

Entities can have child elements and you can independently map each of them to individual parts of the utterances. You also have support for composite entities, which are a collection of characteristics that collectively build up to a single entity.

For better understanding, an entity called "Vehicles" can have child entities called "Cars" and "SUVs," among other names. This relationship can help you map multiple entities into a larger category. In case of "Composite Entities," the individual parts would denote one single entity with various properties of it. An example for a composite entity called Car is 2016 Black Ford Mustang, made up of year, color, make and model information.

Pre-Built Entities

Similar to data types in any programming language, the LUIS service includes a set of entities for a number of common entities, so you don't have to go out and think about every possible term that your users may throw at you. Some examples include most common variations of date, time, numbers and geographical entities. You can include the pre-built entities in your application and use them in your labeling activities. Keep in mind that the behavior of pre-built entities cannot be changed.

An exhaustive list of pre-built entities can be found in the LUIS documentation.

While it's possible to add numerous intents and entities (and pre-built entities) in your model, a word of caution is to keep it simple and precise. You can start with the most obvious utterances and add more of them to make it more natural for your users. But keep in mind that thinking ahead of the experience you want to build goes a long way in enhancing the UX and evolving the experience further. If you don't plan ahead and change intents or entities in your models, you'll have to label all the utterance and train your model all over again.

Let's Go Build Something!

It's time to build something and take a ride with LUIS. In this article, I'll look at an inventory application. I'll build a language model using the intents and entities, train the model, and then use this in a bot powered by the Bot Framework and a Universal Windows Platform (UWP) app using the REST endpoint that LUIS exposes for me to use all its capabilities.

To keep it simple, we're going to deal with an inventory of clothes. First, log on to the LUIS portal at luis.ai and create a new application.

This being an inventory application, you'll use it to track inventory of stocks and for that purpose, the first intent that you're going to add is ListItems. You'll map this intent to all the utterances where the user's intent is to query the inventory as a whole or for an item.

When you're creating an intent, the LUIS application will also ask for an example utterance. This utterance is the first natural language query to trigger this intent. Click on the "+" button next to "Intents" and add the name of the intent as "ListItems." You'll keep the example utterance simple: "Show me the inventory."

Saving this intent takes you to the "new utterance" screen. **Figure 1** shows the example utterance along with the ListItems intent mapped to it within the dropdown menu next to it.

Click on the Submit button to add this utterance to your LUIS application. Before LUIS can start working its magic for you, you must add more such utterances to help LUIS understand the intents more accurately. Keep in mind that the utterances must project the same intent as the one in **Figure 1**, but at the same time, they should be something that users will say naturally when asking for stocks: "Show me the stocks" comes to mind.

Now that you've added two utterances to your application, click on the Train button in the lower-left corner to see if the LUIS application has enough information to understand when to trigger the ListItems intent. The framework triggers the training periodically on its own, as well. Training your model after adding a few examples can help you identify any problem with the model early and take corrective actions. Because LUIS framework also features active learning, you'll benefit from training as the example utterance will be scored automatically for you as you add them.

Moving forward with your application, it's also natural to ask about the inventory of certain items you're stocking, so also think about examples such as "Show me the stocks of red scarves" and "How many shirts do we have in stock?"

However, these queries are different from the ones you've added so far. These queries contain the terms "red scarves" and "shirts." This means you need more than your intent, ListItems, to return the right results back to your users. You need an entity called "Item," which you'll map to these terms to add more intelligence in your language model.

You can add these utterances to your LUIS application and label the entities later, but in this case, you'll add entity first and then the utterances. Click on the "+" button next to Entities and name this entity Item.

Now, you can add those queries mentioned earlier and label them with the intent and entity at the same time. To do that, just add your utterance and if the intent hasn't already been mapped with a score, select the relevant intent and then click on the term "shirts" to map it with the Item entity.

Select Item from the list to label it an item. Add the other example already mentioned—"Show me the stocks of red scarves"—and instead of mapping just "scarves," select "red" and "scarves," both as the entity Item. Note: A favorite browser, Edge, doesn't let you select multiple words in the LUIS portal. Use any other browser of your choice to do this.

Also note that the term "red scarves" falls in the category of Composite Entities because they denote one single entity together—scarves, which have red in them. As explained earlier, Composite Entities are made up of multiple parts but represent one unit of object such as "black shoes" and "2016 Ford Mustang." However, for the sake of simplicity, you're going to treat them as a single entity.

Train the model again and see if the active learning in LUIS kicks in. Now try adding an utterance such as, "How many wallets do we have in stock," or, "Show me the stocks of trousers."

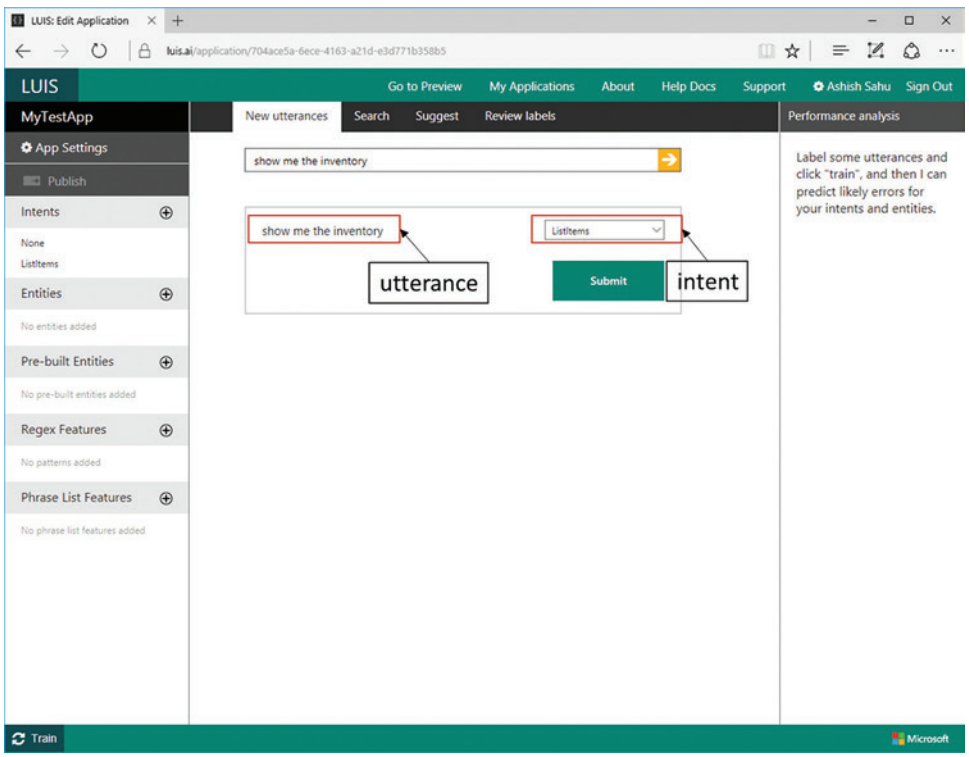


Figure 1 **Example of Utterance and Intent**

You might find the result interesting. Notice that the term "wallets" gets mapped to Item entity but "trousers" doesn't. Don't panic, it just means that LUIS needs a few more examples to make sense of utterances that follow the same pattern. To do that, map "trousers" to Item entity and train your model one more time.

To test this, try adding "Show me the stocks of red shirts" or "Show me the stocks of pants" and verify that red shirts and pants get mapped to the right intents and entities. I hope your mileage doesn't vary from mine so far.

Using the Suggest section in the portal, you can also get suggestions from the Cortana logs for individual intents and entities.

Once your intents and entities are getting mapped correctly, you can move on to the next phases of your journey on LUIS.

BUSINESS FILE APIS

TRY RISK FREE - 30 Day Trial

Open, Create, Convert, Print and Save files from your apps!

Aspose.Total

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

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

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

Aspose.Slides
PPT, POT, ODP, XPS, HTML
PNG, PDF...

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

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

Aspose.Email
MSG, EML, PST, MHT
OST, OFT...

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

+ more!

.NET | Java | Cloud

Download FREE trial at www.aspose.com.

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@aspose.com

ASPOSE
File Format APIs

Figure 2 **Testing the LUIS App**

```
{
  "query": "how many ties do we have in the stock?"
  "intents": [
    {
      "intent": "ListItems",
      "score": 0.9999995
    },
    {
      "intent: "None",
      "score": 0.0582637675
    }
  ],
  "entities": [
    {
      "entity": "ties",
      "type": "Item",
      "startIndex": 9,
      "endIndex": 12,
      "score": 0.903107
    }
  ]
}
```

Using LUIS with Real Apps

This LUIS application isn't useful for your users now; you need to connect to this application from your apps and services. Because the LUIS application is exposed via REST endpoints and the responses are returned in JSON format, you can use LUIS services from any platform or programming language that can connect using HTTPS protocol and parse JSON responses.

Note: The LUIS portal also exposes the export functionality from the "My Application" portion, which exports your LUIS application as a JSON document to make changes offline and import it back. In combination with the LUIS APIs and the C# SDK, you can integrate LUIS in your DevOps processes, as well.

You also need to publish your LUIS app before you can start calling it from your apps, which is as simple as it gets: Just click on the Publish button and click again on the Publish Web service button.

Notice that the REST endpoint URI also includes your LUIS application ID and the subscription key. Protect this information as much as you would any other credentials as it can otherwise lead to disruption of the service and have a financial impact.

Once the application has been published, you should be able to test it by typing any other example in the Query input box and test the accuracy of your model. Try that by entering "how many ties do we have in the stock?" and press Enter on your keyboard.

This will open a new browser window and you should get a response in the JSON format as shown in **Figure 2**.

The response includes the query string passed to the LUIS app, along with the intents and entities detected in the query. Also included is the individual scoring information for each of them. These scores are important because they're direct indicators of how your language model and the training are performing. As you add more utterance and make any changes to your model, this dialog box also provides you with an option to publish your updates. Updating your LUIS application after every training session is important because it'll keep using the older training model and the response from the HTTP endpoint will defer from your expectations.

Figure 3 **Classes to Deserialize the JSON Response**

```
public class LUISResponse
{
  public string query { get; set; }
  public 1Intent[] intents { get; set; }
  public 1Entity[] entities { get; set; }
}

public class 1Intent
{
  public string intent { get; set; }
  public float score { get; set; }
}
public class 1Entity
{
  public string entity { get; set; }
  public string type { get; set; }
  public int startIndex { get; set; }
  public int endIndex { get; set; }
  public float score { get; set; }
}
```

Analyzing Performance of Language Model

Adding too many variations of the language can result in errors and might force you to change your language model. To address these issues, the LUIS portal features a Performance Analysis section. You can use this section to understand how your LUIS app is performing when it comes to detecting intents and entities. You can get a color-coded performance overview of all of your intents and entities in this section.

Depending on the training, examples, and language model used, your LUIS app might also run into issues where it's unable to map intents or entities correctly. There might also be cases where adding multiple types of utterance confuses the LUIS service. These issues can be easily tracked with the performance drill-down using Performance Analysis. The dropdown menu also lets you drill down on analysis to individual intent and entities.

You can also get similar information for the entities in your language model.

This information, along with the Review Labels section of the portal, can help you look at and analyze any errors with your language model.

Calling LUIS From C# UWP/ASP.NET Apps

If you're building a UWP app or ASP.NET Web app using C#, you can use the classes denoted in **Figure 3** to deserialize the JSON response.

The code in **Figure 4** in your C# UWP or ASP.NET app can use these classes to get the intent and entities information.

Based on your requirements, you can run the response through a loop to extract multiple entities of different types, as well as score information about the intents detected in the query string.

Using LUIS with Bot Framework

If you're using Bot Framework to build a bot and are looking to use LUIS to add natural language intelligence, you'll be pleased to know that the Microsoft.Bot.Builder namespace in the Bot SDK makes it extremely easy to connect with your LUIS application and filter out the intents and entities. In the MessageController of your Bot Framework solution, add the following line to route all incoming messages to the class called LuisConnect:

```
await Microsoft.Bot.Builder.Dialogs.Conversation.SendAsync(activity,
  () => new LuisConnect());
```

Now add a class file called LuisConnect.cs in your project and change the code, as shown in **Figure 5**.

Data Quality Made Easy.
Your Data, Your Way.

Melissa Data 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.

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**.

Microsoft® SQL Server® Integration Services

Microsoft Dynamics® CRM

Microsoft® .NET

Microsoft Azure

Microsoft® Excel

Start Your Free Trial

www.MelissaData.com/msft-pd

Germany
www.MelissaData.de

United Kingdom
www.MelissaData.co.uk

India
www.MelissaData.in

Australia
www.MelissaData.com.au

MELISSA DATA®
Your Partner in Global Data Quality

www.MelissaData.com | 1-800-MELISSA

Run your bot locally and try asking questions such as, "Show me the stocks of shirts," or, "How many belts do we have in stock?" and you should get the appropriate responses with the intents and entities back from the bot.

The most interesting part about the code in **Figure 5** is that you just had to label your methods with [LuisIntent] and the SDK takes care of calling the LUIS application and getting back results from the LUIS service. This makes it really quick and simple to start adding the language intelligence in our apps.

Making It Better

The focus of this article is to make you familiar with the workings of LUIS and integration so I've used really simple examples. There are two more features of LUIS that are bound to make your life easier: Regex and Phrase List features.

Much like the name suggests, the Regex feature helps in matching a repetitive pattern in your phrases such as product codes. The Phrase List feature can be used as an interchangeable list of words or phrases to look for in your utterances. For example, in the application we have utterances that started with "Show me the stocks," "Find me the stocks," "How many," and so on. Adding these phrases in a Phrase List called InventoryQueries at the start will remove the need to train your model with more examples for these utterances separately. I'll leave that to you to explore and experience.

The Future

The LUIS offering is ready to be used in your apps but it's still being improved and new features are being added frequently. There are some features that aren't covered in this portal but are available for public preview. They're exciting and still in development:

- **Integration with Bot Framework and Slack:** You can try this out when publishing your LUIS app in Preview Portal. This integration lets you quickly integrate LUIS with Microsoft Bot Framework and Slack.
- **Dialog Support:** Dialog support in LUIS lets you add conversational intelligence in your LUIS application so it can ask for more information from the users on its own if the query requires more information than provided by the users at first. For example, a flight app can prompt for a travel date if the user asks for flight information with just the city name
- **Action Fulfillment:** This feature lets you fulfill the user-triggered actions using the built-in and custom channel right from your LUIS app.

These features are exciting and enable more natural conversational interaction in your app with little effort. They need depth exploration on their own and I hope to do that soon.

Wrapping Up

I hope you now understand what LUIS can do for you and how effortlessly you can start leveraging it to add a more natural human interaction element to your apps.

In this article, I went through the basics of the LUIS service. I created a LUIS application, built and trained your language model to help you understand what users mean when they ask something. I also looked at the ways in which this LUIS application can be used from your apps, Web services and in your bots. A sample project that contains the LUIS model, UWP app and the bot sample code mentioned in this article can be found on GitHub at bit.ly/2eEmPsy. ■

Figure 4 **Code Used to Get Intent and Entities Information**

```
Private async Task LUISParse(string queryString)
{
  using (var client = new HttpClient())
  {
string uri =
  https://api.projectoxford.ai/luis/v1/application?id=<application-
id>&subscription-key=<subscription-key>&q= + queryString;
HttpResponseMessage msg = await client.GetAsync(uri);

if (msg.IsSuccessStausCode)
{
  var jsonresponse = await msg.Content.ReadAsStringAsync();
  var _Data =
    JsonConvert.DeserializeObject<LUISResponse>(jsonresponse);

  var entityFound = _Data.entities[0].entity;
  var topIntent = _Data.intents[0].intent;
}

}
```

Figure 5 **Adding the Class File LuisConnect.cs**

```
using System;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;
using Microsoft.Bot.Connector;

namespace BotApp2
{
  [LuisModel("<application-id>", "<subscription-key>")]
  [Serializable]
  public class Luis: LuisDialog<object>
  {
    [LuisIntent("")]
    public async Task None(IDialogContext context, LuisResult result)
    {
      stringmessage =
        "I'm sorry I didn't understand. Try asking about stocks or inventory.";
        await context.PostAsync(message);
        context.Wait(MessageReceived);
    }

    [LuisIntent("ListItems")]
    public async Task ListInventory(IDialogContext context, LuisResult result)
    {
      string message = "";
      if (result.Entities.Count !=0 && result.Intents.Count 0 )
        message = $ "detected the intent \ "{result.Intents[0].Intent}\"
        for \"{result.Entities[0].Entity}\". Was that right?";
      await context.PostAsync (message);
      conext.Wait(MessageReceived);
    }

    public async Task Start Async(IDialogContext context)
    {
      context.Wait(MessageReceived);
    }
  }
}
```

Ashish Sahu *is a senior technical evangelist, working with Developer Experience at Microsoft India, and helping ISVs and startups overcome technical challenges, adopt latest technologies, and evolve their solutions to the next level. He can be contacted at ashish.sahu@microsoft.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Srikantan Sankaran*

We Made WPF GREAT!

Xceed
Business Suite
for WPF

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

Xceed

Introduction to the HoloLens, Part 2: Spatial Mapping

Adam Tuliper

In my last article, I talked about the three pillars of input for the HoloLens—gaze, gesture and voice (msdn.com/magazine/mt788624). These constructs allow you to physically interact with the HoloLens and, in turn, the world around you. You're not constrained to working only with them, however, because you can access information about your surroundings through a feature called spatial mapping, and that's what I'm going to explore in this article.

If I had to choose a single favorite feature on the HoloLens, it would be spatial mapping. Spatial mapping allows you to understand the space around you, either explicitly or implicitly. I can explicitly choose to work with the information taken in, or I can proceed implicitly by allowing natural physical interactions, like dropping a virtual ball onto a physical table, to take place. Recently, with some really neat updates to the HoloToolkit from Asobo Studio, it's easy to scan for features in your environment, such as a chair, walls and more.

This article discusses:

- What a 3D model is
- How spatial mapping works
- Interacting with the spatial mesh

Technologies discussed:

HoloLens, HoloToolkit, Visual Studio, Unity, C#

What Is a 3D Model?

It might be helpful to understand what a 3D model is before looking at what a spatial map of your area represents. 3D models come in a number of file formats, such as .ma or .blender, but often you'll find them in either of two proprietary Autodesk formats called .FBX (Filmbox) or .OBJ files. .FBX files can contain not only 3D model information, but also animation data, though that isn't applicable to this discussion.

A 3D model is a fairly simple object, commonly tracked via face-vertex meshes, which means tracking faces and vertices. For nearly all modern hardware, triangles are used for faces because triangles are the simplest of polygons. Inside a 3D model you'll find a list of all vertices in the model (made up of x,y,z values in space); a list of the vertex indices that make up each triangle; normals, which are just descriptive vectors (arrows) coming off each vertex used for lighting calculations so you know how light should interact with your model; and, finally, UV coordinates—essentially X,Y coordinates that tell you how to take a 2D image, called a texture, and wrap it around your model like wrapping paper to make it look like it was designed. **Figure 1** shows virtual Adam, a model that the company xxArray created for me because, well, I wanted to put myself into a scene with zombies. This is just a 3D model, but note the legs, which are made of vertices and triangles, and that the pants texture is, in simple terms, wrapped around the 3D model of the legs to look like pants. That's nearly all the magic behind a 3D model.

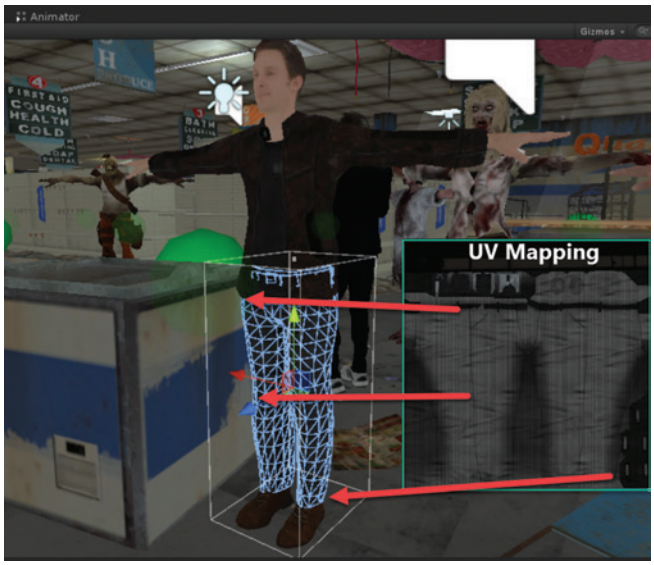


Figure 1 **UV Mapping of 2D Texture to 3D Object**

What Does Spatial Mapping Look Like?

Spatial mapping is easier in some ways because you're not dealing with the textures of your environment. All you typically care about is having a fairly accurate mesh created from your environment that can be discovered. The environment is scanned so you can interact with it. **Figure 2** shows a scenario slightly more like what you'll actually get, though contrived. The model on the left shows the vertices, triangles and normals. You can't see the normal directly, of course, but you see its result by how the object is shaded.

What you've seen thus far in both 3D model scenarios is purely for rendering and has absolutely nothing to do (yet) with physics. The green box outline on the right in **Figure 2** is the shape of the collider I've moved off the cube to show a point; this is the component that defines the region to the physics system. If you want to fully interact with the world on the HoloLens, a game or in any 3D experience, really, you need a collider for the physics system to use.

When you turn the HoloLens on and are in the holographic shell, it's always mapping your environment. The HoloLens does this to understand where to place your windows. If I walk around my house with the HoloLens, it's always updating its information about my environment. This serves two purposes: First, when I walk into a room I've been in previously, the HoloLens should show me the windows I had open. Second, environments are always changing and it needs to detect those changes. Think of the following common scenarios: someone walks in front of me, my kids are running around in the house, our pet bear walks by and creates a large occlusion zone I can't see through. The point is, the environment is potentially always changing and the HoloLens is looking for these changes. Before delving into the API, let's see the spatial mapping in practice (and, by the way, I don't have a real pet bear).

To view spatial mapping in action, you can connect to the Windows Device Portal on a HoloLens, which allows remote management and viewing of the device, including a live 30 FPS video stream of what the device sees. The device portal can be run for nearly any Windows 10 device. It can be accessed by going to the device IP, or to 127.0.0.1:10080 for devices plugged in over USB once

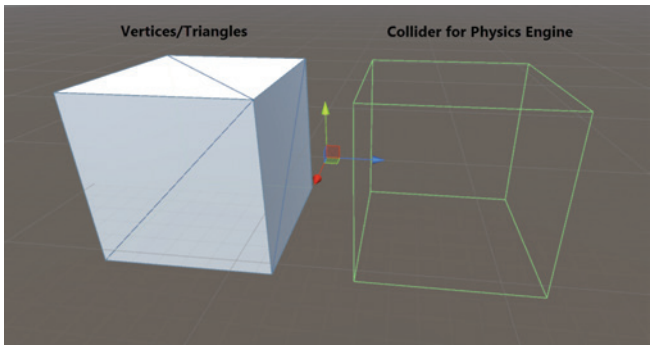


Figure 2 **What's Needed for Rendering and for the Physics Engine**

it's been enabled on the HoloLens in the Developer Settings. Most Windows 10 devices can be enabled for a device portal as outlined at bit.ly/2f0cnfM. **Figure 3** and **Figure 4** show the spatial mesh retrieved from the 3D view in the device portal. **Figure 3** shows what the HoloLens sees as soon as I turn it on, while **Figure 4** displays the view after a brief walk through my living room.

How Spatial Mapping Works

Spatial mapping works via a SurfaceObserver object, as you're observing surface volumes, watching for new, updated and removed surfaces. All the types you need to work with come with Unity out of the box. You don't need any additional libraries, though the HoloToolkit-Unity repository on GitHub has lots of functionality for the HoloLens, including some amazing surface detection I'll look at later, so this repository should be considered essential for hitting the ground running.

First, you tell the SurfaceObserver that you're observing a volume:

```
public Vector3 Extents = new Vector3(10, 10, 10);
observer = new SurfaceObserver();
// Start from 0,0,0 and fill in a 10 meter cube volume
// as you explore more of that volume area
observer.SetVolumeAsAxisAlignedBox(Vector3.zero,Extents);
```

The larger the region, the greater the computational cost that can occur. According to the documentation, spatial mapping scans in a 70-degree cone a region between 0.8 and 3.1 meters—about 10 feet

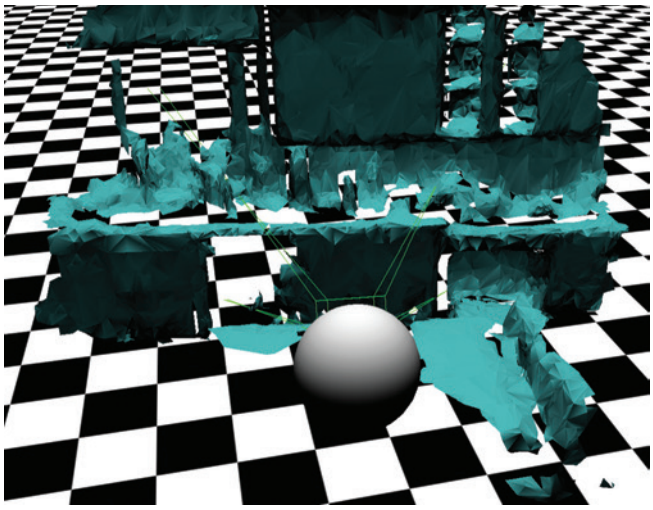


Figure 3 **HoloLens Spatial Mesh Right After HoloLens Is Turned on in a New Room**

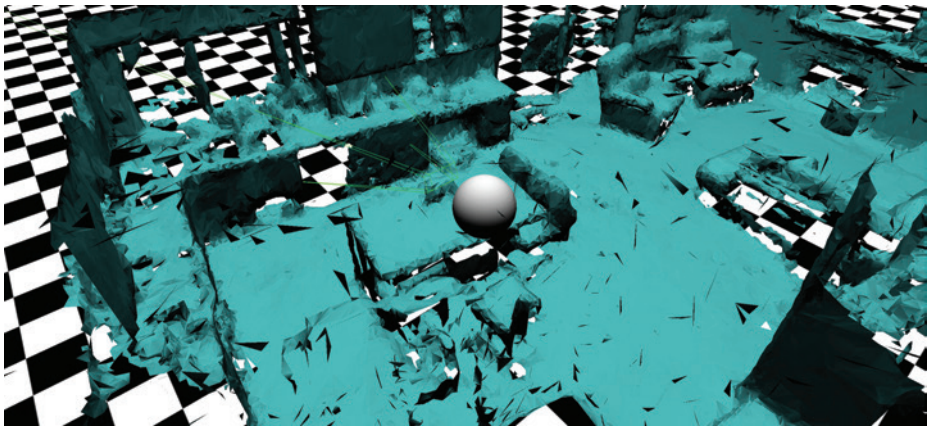


Figure 4 **HoloLens Spatial Mesh After a Quick Walk-Through a Portion of the Room**

out (the docs state these values might change in the future). If an object is further away, it won't be scanned until the HoloLens gets closer to it. Keeping to 0.8 meters also ensures the user's hands won't accidentally be included as part of the spatial mesh of the room.

The process to get spatial data into an application is as follows:

1. Notify the SurfaceObserver to observe a region of size A and shape B.
2. At a predefined interval (such as every 3 seconds), ask the SurfaceObserver for an update if you aren't waiting on other results to be processed. (It's best not to overlap results; let one mesh finish before the next is processed.)
3. Surface Observer lets you know if there's an add, update or removal of a surface volume.
4. If there's an add or update to your known spatial mesh:
   a. Clean up old surface if one exists for this id.
   b. Reuse (to save memory, if you have a surface that isn't being used) or allocate a new SurfaceObject with mesh, collider and world anchor components.
   c. Make an async request to bake the mesh data.
5. If there's a removal, remove the volume and make it inactive so you can reuse its game object later (this prevents additional allocations and thus fewer garbage collections).

To use spatial mapping, SpatialPerception is a required capability in a Universal Windows Platform (UWP) app. Because an end user should be aware that an application can scan the room, this needs to be noted in the capabilities either in the Unity player settings as shown in **Figure 5**, or added manually in your application's package.appxmanifest.

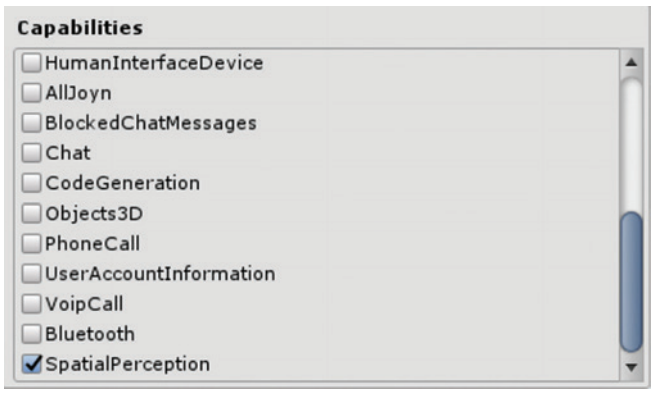


Figure 5 **Adding SpatialPerception in File-Build Settings**

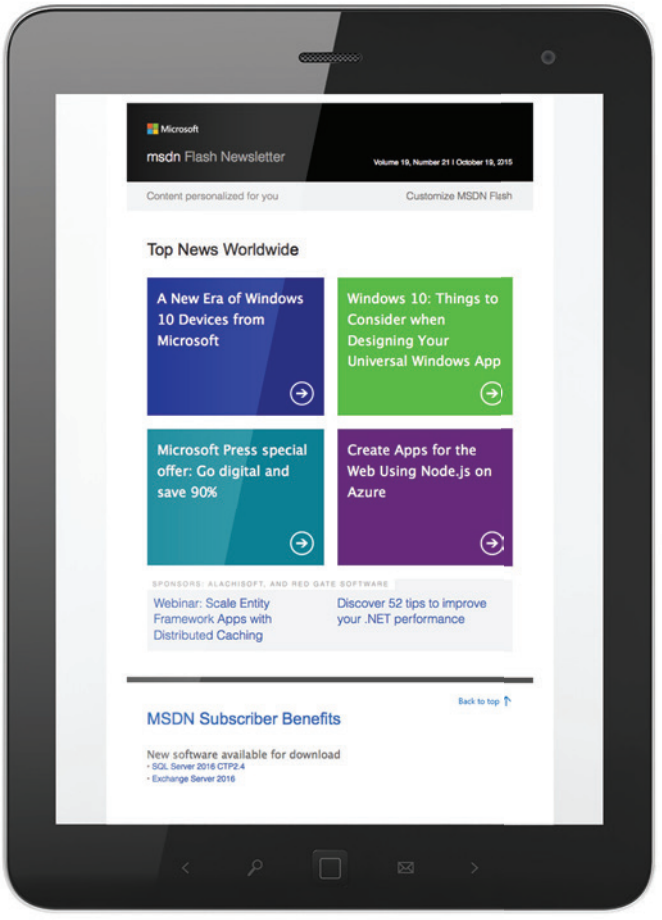
The spatial meshes are processed in surface volumes that are different from the bounding volume defined for the SurfaceObserver to observe. The key is once the SurfaceObserver_OnSurface delegate is called to note surface volume changes, you request the changes in the next frame. The meshes are then prepared in a process called baking, and a SurfaceObserver_OnDataReady callback is processed when the mesh is ready.

Baking is a standard term in the 3D universe that usually refers to calculating something ahead of time. It's typically used to talk about calculating lighting information and transferring it to a special image called a lightmap in the baking process. Lightmaps help avoid runtime calculations. Baking a mesh can take several frames from the time you ask for it in your Update function (see **Figure 6**). For performance's sake, request the mesh only from RequestMeshAsync if you're actually going to use it, otherwise you're doing extra processing when you bake it for no reason.

The Update code is called every frame on any game object deemed responsible for getting the spatial meshes.

When surface volume baking is requested via RequestMeshAsync, the request is passed a SurfaceData structure in which you can specify the scanning density (resolution) in triangles per cubic meter to process. When TrianglesPerCubicMeter is greater than 1000, you get fairly smooth results that more closely match the surfaces you're scanning. On the other hand, the lower the triangle count, the better the performance. A resolution of <100 is very fast, but you lose surface details, so I recommend trying 500 to start and adjusting from there. **Figure 7** uses about 500 TrianglesPerCubicMeter. The HoloLens already does some optimizations on the mesh, so you'll need to performance test your applications and make a determination whether you want to scan and fix up more (use less memory) or just scan at a higher resolution, which is easier but uses more memory.

Creating the spatial mesh isn't a super high-resolution process by design because higher resolution equals significantly more processing power and usually isn't necessary to interact with the world around you. You won't be using spatial mapping to capture a highly detailed small figurine on your countertop—that's not what it's designed for. There are plenty of software solutions for that, though, via a technique called photogrammetry, which can be used for creating 3D models from images, such as Microsoft 3D Builder, and many others listed at bit.ly/2fzcH1z and bit.ly/1UjAt1e. The HoloLens doesn't include anything for scanning and capturing a textured 3D model, but you can find applications to create 3D models on the HoloLens, such as HoloStudio, or you can create them in 3D Builder (or in any 3D modeling software for that matter) and bring them into Unity to use on the HoloLens. You can also now live stream models from Unity to the HoloLens during development with the new Holographic emulation in Unity 5.5.



Get news from MSDN in your inbox!

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

msdn magazine

msdn.microsoft.com/flashnewsletter

Figure 6 **The Update Function**

```
private void Update()
{
  // Only do processing if you should be observing.
  // This is a flag that should be turned on or off.
  if (ObserverState == ObserverStates.Running)
  {
    // If you don't have a mesh creation pending but you could
    // schedule a mesh creation now, do it!
    if (surfaceWorkOutstanding == false && surfaceWorkQueue.Count > 0)
    {
      SurfaceData surfaceData = surfaceWorkQueue.Dequeue();
      // If RequestMeshAsync succeeds, then you've scheduled mesh creation.
      // OnDataReady is left out of this demo code, as it performs
      // some basic cleanup and sets some material/shadow settings.
      surfaceWorkOutstanding = observer.RequestMeshAsync(surfaceData,
        SurfaceObserver_OnDataReady);
    }
    // If you don't have any other work to do, and enough time has passed since
    // previous update request, request updates for the spatial mapping data.
    else if (surfaceWorkOutstanding ==
      false && (Time.time - updateTime) >= TimeBetweenUpdates)
    {
      // You could choose a new origin here if you need to scan
      // a new area extending out from the original or make Extents bigger.
      observer.SetVolumeAsAxisAlignedBox(observerOrigin, Extents);
      observer.Update(SurfaceObserver_OnSurfaceChanged);
      updateTime = Time.time;
    }
  }
}
private void SurfaceObserver_OnSurfaceChanged(
  SurfaceId id, SurfaceChange changeType, Bounds bounds, System.DateTime updateTime)
{
  GameObject surface;
  switch (changeType)
  {
    case SurfaceChange.Added:
    case SurfaceChange.Updated:
      // Create (or get existing if updating) object on a custom layer.
      // This creates the new game object to hold a piece
      // of the spatial mesh.
      surface = GetSurfaceObject(id.handle, transform);

      // Queue the request for mesh data to be handled later.
      QueueSurfaceDataRequest(id, surface);
      break;

    case SurfaceChange.Removed:
      // Remove surface from list.
      // ...
      break;
  }
}
```

Mesh colliders in Unity are the least-performant colliders, but they're necessary for surfaces that don't fit primitive shapes like boxes and spheres. As you add more triangles on the surfaces and add mesh colliders to them, you can impact physics performance. SurfaceData's last parameter is whether to bake a collider:

```
SurfaceData surfaceData = new SurfaceData(id,
            surface.GetComponent<MeshFilter>(),
            surface.GetComponent<WorldAnchor>(),
            surface.GetComponent<MeshCollider>(),
            TrianglesPerCubicMeter,
            bakeCollider);
```

You may never need a collider on your spatial mesh (and thus pass in bakeCollider=false) if you only want to detect features in the user's space, but not integrate with the physics system. Choose wisely.

There are plenty of considerations for the scanning experience when using spatial mapping. Applications may opt not to scan, to scan only part of the environment or to ask users to scan their environment looking for certain-size surfaces like a couch. Design guidelines are listed on the "Spatial Mapping Design" page of the Windows Dev Center (bit.ly/2gDqQQi) and are important to consider, especially because understating scenarios can introduce various imperfections into your mesh, which fall into three general categories discussed on the "Spatial Mapping Design" page—bias, hallucinations and holes. One workflow would be to ask the user to scan everything up front, such as is done at the beginning of every "RoboRaid" session to find the appropriate surfaces for the game to work with. Once you've found applicable surfaces to use, the experience starts and uses the meshes that have been provided. Another workflow is to scan up front, then scan continually at a smaller interval to find real-world changes.

Working with the Spatial Mesh

Once the mesh has been created, you can interact with it in various ways. If you use the HoloToolkit, the spatial mesh has been created with a custom layer attribute. In Unity you can ignore or include layers in various operations. You can shoot an invisible arrow out in a common operation called a raycast, and it will return the colliders that it hit on the optionally specified layer.

Often I'll want to place holograms in my environment, on a table or, even like in "Young Conker" (bit.ly/2f4Ci4F), provide a location for the character to move to by selecting an area in the real world (via the spatial mesh) to which to go. You need to understand where you can intersect with the physical world. The code in **Figure 8** performs a raycast out to 30 meters, but will report back only areas hit on the spatial mapping mesh. Other holograms are ignored if they aren't on this layer.

I don't have to use the spatial mesh, of course. If I want a hologram to show up and the user to be able to place it wherever he wants (maybe it always follows him) and it will never integrate with the physical environment, I surely don't need a raycast or even the mesh collider.

Figure 7 **A Virtual Character Detecting and Sitting on a Real-World Item (from the Fragments Application)**

Now let's do something fun with the mesh. I want to try to determine where in my living room an area exists that a character could sit down, much like the scene in **Figure 7**, which is from "Fragments," an amazing nearly five-hour mystery-solving experience for the HoloLens that has virtual characters sitting in your room at times. Some of the code I'll walk through is from the HoloToolkit. It came from Asobo Studio, which worked on "Fragments." Because this is mixed reality, it's just plain awesome to develop experiences that mix the real world with the virtual world.

The entire code example for this is in the HoloToolkit, but let's walk through the process. I've trimmed down the code into applicable pieces. (I've talked about SurfaceObserver already so that will be excluded from this section.) SpatialUnderstandingSourceMesh wraps the SurfaceObserver through a SpatialMappingObserver class to process meshes and will create the appropriate MeshData objects to pass to the SpatialUnderstaing DLL. The main force of this API lies in this DLL in the HoloToolkit.

In order to look for shapes in my spatial mesh using the DLL, I must define the custom shape I'm looking for. If I want a sittable surface that's between 0.2 and 0.6 meters off the floor, made of at least one discrete flat surface, and a total surface area minimum of 0.2 meters, I can create a shape definition that will get passed to the DLL through AddShape (see **Figure 9**).

Next, I can detect the regions and then visualize or place game objects there. I'm not limited to asking for a type of shape and getting all of them. If I want, I can structure my query to QueryTopology_FindLargePositionsOnWalls or QueryTopology_FindLargestWall, as shown in **Figure 10**.

There's also a solver in the HoloToolkit that allows you to provide criteria, such as "Create 1.5 meters away from other objects":

```
List<ObjectPlacementRule> rules =
new List<ObjectPlacementRule>() {
    ObjectPlacementRule.Create_AwayFromOtherObjects(1.5f),
};
// Simplified api for demo purpose - see LevelSolver.cs in the HoloToolkit.
var queryResults = Solver_PlaceObject(....)
```

After executing the preceding query to place an object, you get back a list of results you can use to determine the location, bounds and directional vectors to find the orientation of the surface:

```
public class ObjectPlacementResult
{
  public Vector3 Position;
  public Vector3 HalfDims;
  public Vector3 Forward;
  public Vector3 Right;
  public Vector3 Up;
};
```

Figure 8 **Performing a Raycast**

```
// Do a raycast into the world that will only hit the Spatial Mapping mesh.
var headPosition = Camera.main.transform.position;
var gazeDirection = Camera.main.transform.forward;

RaycastHit hitInfo;
// Ensure you specify a length as a best practice. Shorter is better as
// performance hit goes up roughly linearly with length.
if (Physics.Raycast(headPosition, gazeDirection, out hitInfo,
    10.0f, SpatialMappingManager.Instance.LayerMask))
{
  // Move this object to where the raycast hit the Spatial Mapping mesh.
    this.transform.position = hitInfo.point;

  // Rotate this object to face the user.
  Quaternion rotation = Camera.main.transform.localRotation;
  rotation.x = 0;
  rotation.z = 0;
  transform.rotation = rotation;
}
```

Figure 9 **Creating a Shape Definition**

```
ShapeDefinitions.cs
// A "Sittable" space definition.
shapeComponents = new List<SpatialUnderstandingDllShapes.ShapeComponent>()
{
  new SpatialUnderstandingDllShapes.ShapeComponent(
    new List<SpatialUnderstandingDllShapes.ShapeComponentConstraint>()
    {
      SpatialUnderstandingDllShapes.ShapeComponentConstraint.Create_
        SurfaceHeight_Between(0.2f, 0.6f),
      SpatialUnderstandingDllShapes.ShapeComponentConstraint.Create_
        SurfaceCount_Min(1),
      SpatialUnderstandingDllShapes.ShapeComponentConstraint.Create_
        SurfaceArea_Min(0.20f),
  }),
};
// Tell the DLL about this shape is called Sittable.
AddShape("Sittable", shapeComponents);
```

Figure 10 **Querying for a Shape**

```
SpaceVisualizer.cs (abbreviated)

const int QueryResultMaxCount = 512;
private ShapeResult[] resultsShape = new ShapeResult[QueryResultMaxCount];
public GameObject Beacon;

public void FindSittableLocations()
{
  // Pin managed object memory going to native code.
  IntPtr resultsShapePtr =
    SpatialUnderstanding.Instance.UnderstandingDLL.
    PinObject(resultsShape);

  // Find the half dimensions of "Sittable" objects via the DLL.
  int shapeCount = SpatialUnderstandingDllShapes.QueryShape_FindShapeHalfDims(
    "Sittable",
    resultsShape.Length, resultsShapePtr);

  // Process found results.
  for(int i=0;i<shapeCount;i++)
  {
    // Create a beacon at each "sittable" location.
    Instantiate(Beacon, resultsShape[i].position, Quaternion.identity);

    // Log the half bounds of our sittable area.
    Console.WriteLine(resultsShape[i].halfDims.sqrMagnitude < 0.01f) ?
      new Vector3(0.25f, 0.025f, 0.25f) : resultsShape[i].halfDims)
  }
}
```

Wrapping Up

Spatial mapping lets you truly integrate with the world around you and engage in mixed-reality experiences. You can guide a user to scan her environment and then give her feedback about what you've found, as well as smartly determine her environment for your holograms to interact with her. There's no other device like the HoloLens for mixing worlds. Check out HoloLens.com and start developing mind-blowing experiences today. Next time around, I'll talk about shared experiences on the HoloLens. Until then, keep developing! ■

Adam Tuliper *is a senior technical evangelist with Microsoft living in sunny SoCal. He's a Web dev/game dev Pluralsight.com author and all-around tech lover. Find him on Twitter: @AdamTuliper or at adamtuliper.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Jackson Fields*

Automate Complex Deployments with Release Management

Kraig Brockschmidt

The happy phrase, "Ship it," declares that whatever software you're producing, such as mobile apps and their associated back-end services, is now ready for deployment to customers or, as it's said, "into production." But how, exactly, does software get to that point? In my last article in this series (msdn.com/magazine/mt742871), I explored how Build produces the artifacts that feed into the release pipeline. That bundle of artifacts, called the "release," is what then undergoes any number of tests and other processes, as shown in **Figure 1**, on its journey to production. The bulk of DevOps activities, in fact, involve the deployment of a release into any number of environments where certain tests can be run, and shepherding that release along to the next stage of the pipeline.

This article discusses:

- The need for managing the complexity of deployment and testing across different environments and multiple release pipelines
- The notion of a "release" as it relates to build artifacts and the release pipeline
- Continuous deployment as a culture
- Setting up releases for apps and services in Visual Studio Team Services
- Pre-launch app distribution with HockeyApp

Technologies discussed:

Visual Studio, Visual Studio Team Services, Team Foundation Server, Release Management, Release Definitions, Continuous Deployment, HockeyApp, CodePush

A release process potentially involves a large number of different tests that don't necessarily run simultaneously, and which might also require different machines and devices. It might also involve direct approvals by human beings who eat lunch and leave their desks for evenings and weekends. As a result, the time it takes for a release to get through the entire pipeline could easily be a matter of days. Meanwhile, your dev team continues to work their backlog for subsequent releases, committing code to the source repository, thus triggering more builds that produce artifacts that feed into the appropriate pipeline.

Managing the flow of all these artifacts through multiple pipelines can easily become a complex and demanding task, thus the Release Management tools in Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) are an essential part of the Microsoft DevOps stack.

On the surface, release management looks mostly like administrative support, and thus might not be as technically interesting as other parts of the Microsoft DevOps stack. But like everything else in DevOps, release management is fundamentally a practice that begins as a list of steps you can perform manually, in this case to validate that the artifacts from the Build/CI stage are ready to deploy to subsequent environments. Once you have those steps clearly defined—when to deploy artifacts to a particular environment, which tests to run and the criteria for moving the artifacts to the next stage—you can then use tools to incrementally automate those steps.

In many ways, managing a release is a lot like setting up an automated build, except that the output of release management is the

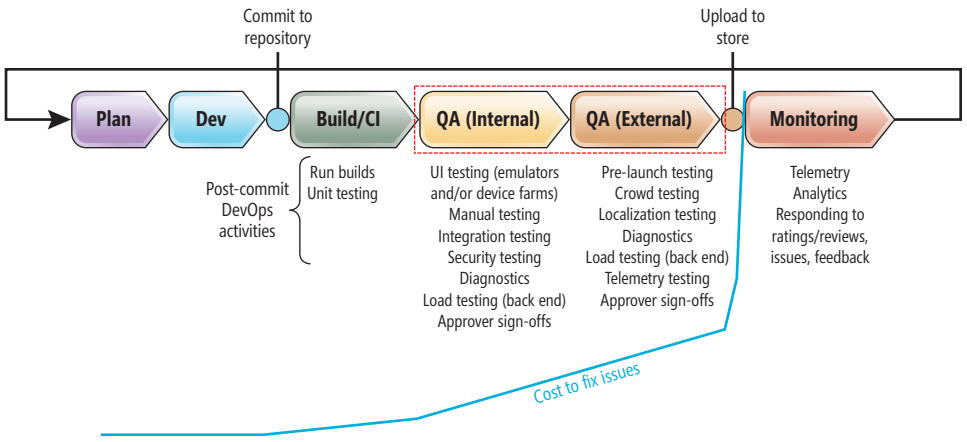


Figure 1 **Managing a Release Happens Between Build and Public Deployment**

deployment of build artifacts to places where the right people can get at them. Ultimately, deployment is how the value contained in your source code is actually delivered to your customers, which is, of course, the whole point of the software development process!

Environments, Pipelines and Managing Complexity

Although the diagram in **Figure 1** shows only two QA stages—one internal and one external—there can really be any number of stages. This is because different forms of testing require deployment into specific environments where those tests can be carried out. An environment is simply a particular configuration of hardware, software and data that's suitable for the desired testing or usage scenarios. Here are a few that are commonly discussed in the context of DevOps:

- Machines that are capable of doing unit tests, integration tests, and UI tests are typically part of an automated test environment, usually using mock data, test services, and servers configured for different load and stress tests.
- A similarly configured manual test environment is generally where a dedicated test team does its work.
- A staging environment is used for deploying apps and services for full-scale pre-production tests (such as upgrade tests), and includes deployment to alpha- and beta-test customers. This environment might draw upon live data and services in a read-only manner, or use staging versions to fully simulate live activity. This is also where you can test your crash and telemetry reporting systems and make sure they're generating the data you want.
- Your public production environment, finally, uses live public data and services, of course, and is where you collect the live telemetry data that will ultimately feed into subsequent releases.

You can, of course, define whatever environments you want according to your validation needs. Whatever the case, how build artifacts travel through these stages and environments—and what tests are applied where—is again what defines a release pipeline, and you can have any number of different routes in operation for different purposes. With mobile apps and back-end services, you'll have multiple deployment targets all along the way—different emulators or device farms, staging servers and so on. You might also have releases go through only part of the pipeline just for testing purposes. With a public release, the "rollout track" feature of Google Play (bit.ly/2b7lh8j) also allows you to release a new version of an app to a limited percentage of customers—which is essentially another production environment—thus, you might have a separate release pipeline for that track.

What Is a Release?

The term "release management" clearly implies that a "release" is being managed in some way. A release is a specific bundle of build artifacts you intend to move through a series of validations and deploy to one or more environments. Those artifacts (along with other information and metadata) always travel through the pipeline as a unit, regardless of any subsequent changes to the source repository and the builds they might trigger, because every set of build artifacts is unique and stamped with a full version number like 1.2.8.12, where the last number is the build. Such versioning buys you complete end-to-end auditing, which is the ability to trace everything that happens within the release pipeline back to the exact build, and thus to an exact set of changes in the source repository.

In DevOps parlance, then, "starting a release" means feeding a worthy unit of build artifacts into the release pipeline. **Figure 2** illustrates what the process might look like: Using the project backlog for communication, the team applies a series of validation checks, irrespective of automation, that either reject the release or allow it to proceed to the next stage, eventually to reach production.

> An environment is simply a particular configuration of hardware, software, and data that's suitable for the desired testing or usage scenarios.

As noted before, you can certainly have any number of testing and staging environments, and even multiple production environments (as when using the Google rollout feature). Because a release process can take considerable time, you might choose to deploy to production only once a week or once a month. In the meantime, you'll still want to run other releases through some part of the pipeline, perhaps deploying to beta testers each week to get feedback for each monthly production release. You'll probably also want daily builds to go to the test manager, and likely want every build to go through a series of quick, automated tests to provide

ongoing feedback to the dev team. In short, the frequency of releases can change as you go up and down the diagram in **Figure 2**.

Thus, release management is something that can start simple and grow from there. The simplest form of this practice, in fact, is something you've likely done already: You build an app package, deploy it to a device (staging), and play with the app to make sure it works as expected. Then you upload the package to a store, hit "Publish"—and presto! You've just done a manual release process to put your app into production.

> A release is a specific bundle of build artifacts you intend to move through a series of validations and deploy to one or more environments.

Continuous Deployment as a Culture

As I explained in the first article in this series, (msdn.com/magazine/mt767694), all DevOps processes begin with being completely clear about what needs to happen at every stage along the release pipeline, automated or not. You should be able to describe all your processes in a simple document, such that every step could be done manually. Then you're ready to apply automation to reduce costs, improve reliability and consistency, and increase frequency of testing and deployment.

A primary goal within DevOps is to have every new release of an app or service—including releases with only minor changes—flow as quickly as possible from the source repository to customers. A completely automated flow is called continuous deployment (CD), which goes hand-in-hand with continuous integration (CI): Every commit to your repository triggers a new CI build, and every successful build—which produces a new bundle of artifacts with a specific version number—triggers a new automated release process. That release process then carries out all the necessary validations before deploying that bundle to production. CD, in short, means continually delivering value to your customers at the lowest cost, with minimal (if any) human intervention along the release pipeline.

Realize, however, that although CD optimizes the release pipeline between the Build/CI stage and deployment to production, it still requires vigilant effort by people to make it work:

- Your team needs strong code-review processes to prevent poor code from being committed to the repository to begin with.
- Your team must have high confidence that automated tests—which people create—are catching most defects and preventing them from reaching customers.
- Because no suite of tests is perfect, some defects *will* get through to production, so your team must actively monitor crash reports, telemetry and direct customer feedback in your production environment.
- Your team must be committed to quickly triaging and prioritizing issues and feeding them into the dev backlog so that corrections quickly get into subsequent releases.
- Issues also identify gaps in your code-review process and test coverage, thus driving improvements in both.

In short, CD isn't just a matter of automating your release pipeline: CD is a culture of using feedback to constantly improve *how* you're delivering value to customers.

As an example, the documentation for Microsoft Azure, found on azure.microsoft.com/documentation, is managed within an open source repository on GitHub, github.com/Azure/azure-content. There's a full CI/CD system in place such that any changes accepted into the repository through pull requests quickly get out to production. Pull requests, however, are carefully scrutinized by the Azure content team at Microsoft, which prevents incorrect or inappropriate edits from getting into the repository at all. Accepted changes then pass through the automated CI/CD pipeline that applies a variety of validation tests (such as catching incorrect formatting and broken links), and then publishes the content to the live site. The team then monitors telemetry reports from Application Insights along with customer comments, using that information

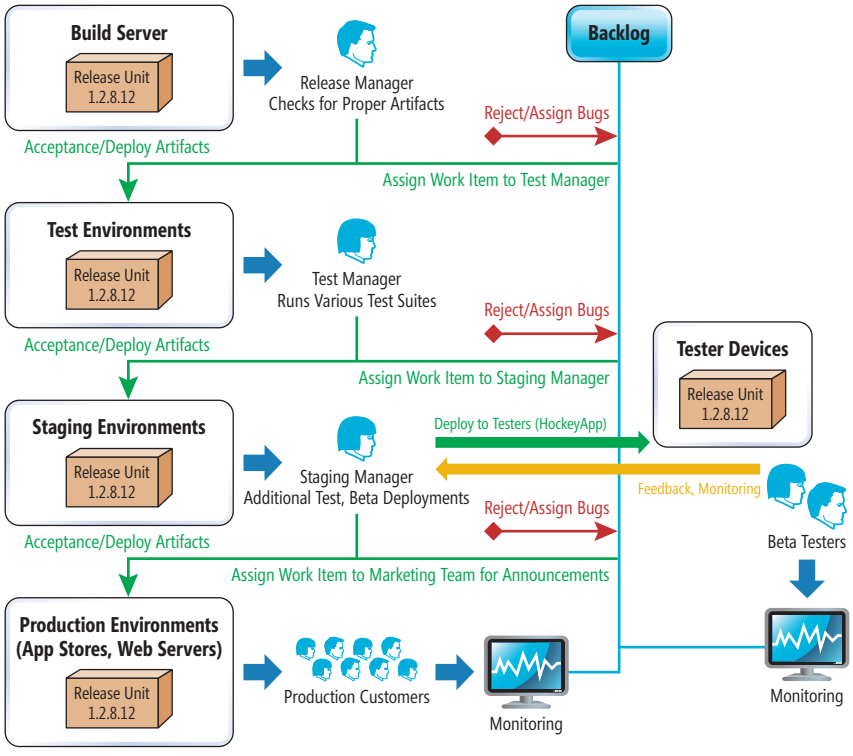


Figure 2 **A Unit of Build Artifacts Flows Through Various Stakeholders on Its Way to Production**

R#

ReSharper

THE LEGENDARY EXTENSION
TO VISUAL STUDIO*

jetbrains.com/msdn

JET BRAINS

THE DRIVE
TO DEVELOP

* WARNING! PROLONGED USE
MAY CAUSE ADDICTION

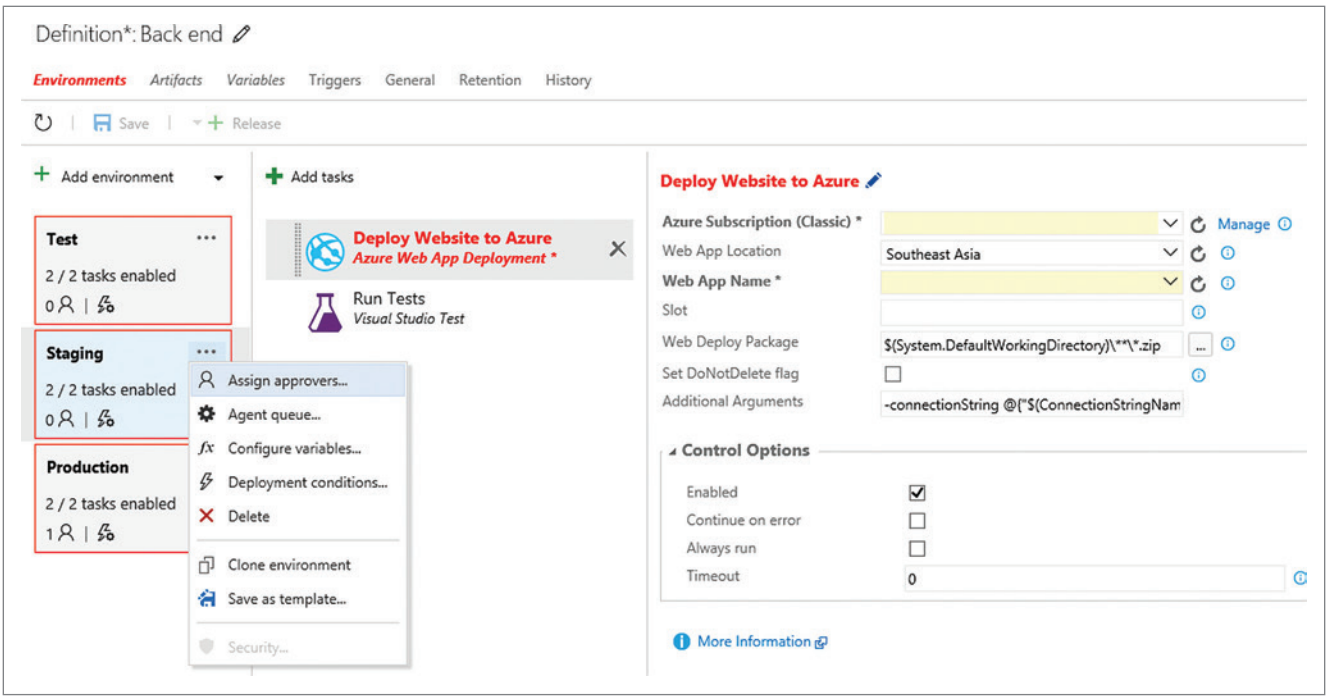


Figure 3 **Editing a Release Definition on Visual Studio Team Services**

to improve the content, improve the validations tests, and improve the review process itself.

Release Management in VSTS

In my last article, I looked at setting up automated builds with VSTS by taking a known build process, creating a build definition from it and feeding that definition to a build agent that's capable of performing the necessary tasks to produce a bundle of artifacts with a specific version number.

> Continuous deployment is a culture of using feedback to constantly improve how you're delivering value to customers.

Release Management in VSTS is a similar process: You create a release definition that specifies how that bundle is to be deployed to different environments, and the tests and validations that are to be applied to it. That release definition is then fed to a release agent—a suitably configured machine—for processing (release agents are managed in the same way as build agents, see bit.ly/2coBQxx). That said, there are a number of significant differences between build and release definitions:

- A build definition produces testable and deployable artifacts; a release definition guides the actual deployment to an environment and the running of tests.
- A build definition always works from a single-source repository; a release definition can draw from any number of build definitions to collect the artifacts for a release.
- You can often set up a fully automated build in a matter of hours; a fully automated release pipeline happens over time because it takes much more effort to create the underlying automated tests and work out the details for approvals and sign-offs.
- A typical build completes in a matter of minutes; a full release process, with multiple environments and manual approval steps, will take much longer. As a result, you'll be monitoring and auditing many releases in your pipeline at different stages.
- A release definition supports both pre- and post-deployment approvers, with which you inject manual control at either end of a release step, as well as explicit manual intervention tasks. A simple example is using one or more pre-deployment approvers before going to production. VSTS also lets you use a group as an approver, such that only one person in that group needs to sign off. In general, you assign an approver any time you want a human being to be involved in the release process, which also results in leaving an audit trail.

As an example, let's say I have a Xamarin app and back-end code in a Git repository on VSTS, with four build definitions that produce the back-end artifacts and app packages for iOS, Android and Windows. A simple release process—echoing the stages shown in **Figure 2**—might be as follows, where a failure at any point in the pipeline will cancel the release:

1. Run unit and integration tests on a successful build.
2. Test environment
   - Deploy the app to Xamarin Test Cloud to run tests on physical devices.
   - Deploy the back end to a local load-test server.
3. Staging environment
   - Deploy the app to beta testers using HockeyApp.
   - Deploy the back end to a staging server (used by beta testers) running on Azure App Service.

4. Require sign-off by an approver who reviews beta-tester feedback from and evaluates the readiness of the release.
5. Production environment
   - Deploy the app to Google Play, Apple App Store and Windows Store.
   - Deploy the back end to the production Azure App Service.

Note, once again, that this process says nothing about automation—it simply describes the steps involved in making a release. Because of this, I can start building release definitions with simple steps like deployment and approver sign-offs. Then, as I get various tests put together, I can incrementally add steps to increase automation. (By the way, this process is essentially what was set up for the MyDriving project [aka.ms/iotsampleapp], although it ends at step 3 because the app is distributed only through HockeyApp.)

Technically speaking, you can include testing and deployment steps directly in a VSTS build definition. This is sufficient when deploying to and testing in only a single environment. When multiple environments, approvers, and other release-specific steps are involved, however, you'll need the fine-grained control of Release Management.

Walk-Through: Deploying to Successive Azure Environments

Now I'll do a walk-through of the main process of using Release Management, for which I've created a Xamarin app and a Node.js service from templates in Visual Studio. In VSTS, I created a Team Project called MSDN Magazine Dec 2016, and added the projects to its source repository. I set up four build definitions to generate suitable artifacts that I can follow through a release pipeline, even if those artifacts don't do anything interesting. (To create build definitions for different back-end project types, see "Build Your App" [bit.ly/2cGbq7W] in the VSTS documentation, which ensures that you create deployable artifacts for Azure and demonstrates deployment tasks in build definitions.)

Because I have four distinct bundles of artifacts going to different targets, I'll eventually need four release definitions, but here I'll start with just the back end. On the Team Services portal, I navigate to the Team Project, select the Release tab, and click + New definition. As with Team Foundation Build, this brings up a dialog with (as of this writing) just a few Azure-related release templates—for any destination other than Azure, including app stores, just start with an empty definition. For my back end, I use the Azure Website Deployment template and click Next. This brings up a configuration dialog in which I select my back end's build definition as the source of the artifacts (you can also use Jenkins as a source). I also select an agent queue and set an option for continuous deployment, which I'll return to later.

VSTS then opens the release definition in the editor shown in **Figure 3** (by default there will be just a single environment). The editor is organized, left to right, into environments, tasks and details for the task. The general workflow is to create an environment first, and then populate it with appropriate tasks for that environment. Because environments often have similar steps, you can create the tasks for one environment and then clone that environment to save time. The + Add Environment button gives you this option. (You can also create meta-tasks, in which you group a sequence of tasks together to create a new single task, parameterized with variables, that can be used in build and release definitions. For a full walk-through, refer to the VSTS documentation at bit.ly/2c1X3fP.)

As with build definitions, any tab that needs attention is highlighted in red—in **Figure 3**, I need to identify the target Azure App Service for the deployment. In my Azure account, then, I create App Service instances called kraigb-MSDN1216-test, kraigb-MSDN1216-staging and kraigb-MSDN1216-prod. I then return to VSTS and click the Manage link on the right-hand side next to Azure Subscription (Classic), which takes me to the Services tab on the Team Project control panel. I select + New Service Endpoint, select Azure Classic, and get a dialog in which I enter connection information. The easiest way to work with this is to select the publish settings file link in the dialog that goes to the Azure portal and generates a text file for download, from which you can copy-paste values into VSTS.

Having established this connection, I return to the release definition, refresh the subscription list and select my new connection. From there I can refresh the Web app location and Web app name controls to select the App Service instance I want.

Because most of the connection information for my Test environment is the same for my Staging and Production environments, I can clone Test twice and rename the copies. When I make the clone for Production, though, I also set myself as a pre-deployment approver and check a box so I get an e-mail when a release is waiting. By doing this, I've injected a pause between the Staging and Production environments.

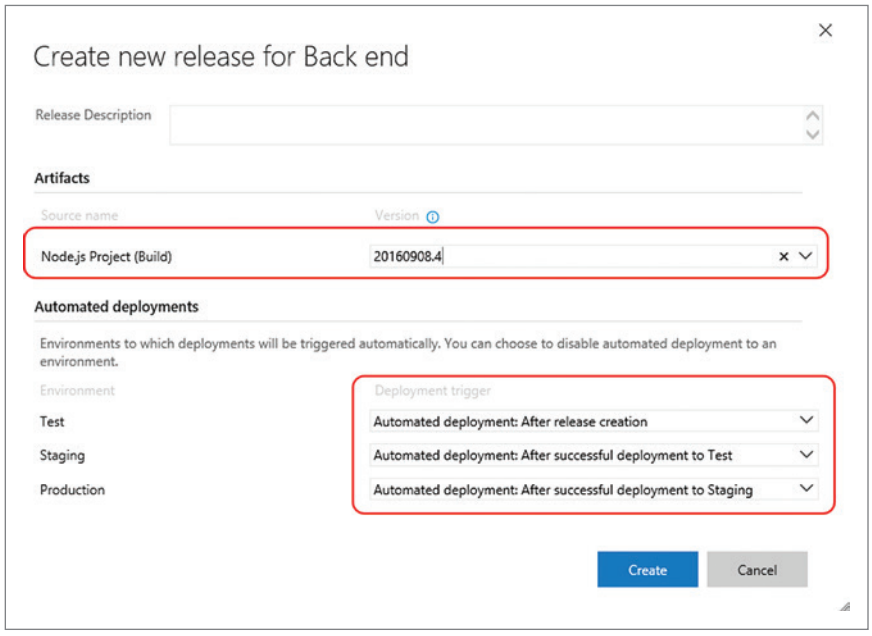


Figure 4 **Starting a New Release Manually by Selecting a Build and Setting Deployments**

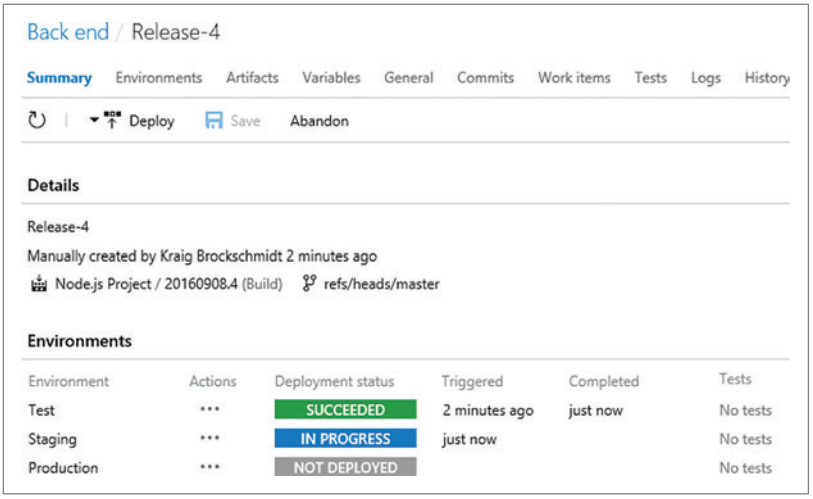


Figure 5 **Examining the Status of a Release in Progress**

see in **Figure 6**. I also get an e-mail alerting me to the approval, with a link to that same release page in VSTS. There I click the Approve or Reject link to indicate my decision, add comments, defer the deployment to a later time or reassign the approval to someone else. In this case I click Approve, the release completes, and I'm able to go see the deployment on the kraigb-1216-prod App Service instance.

Figure 6, as you can see, shows the status for a single release. By clicking on the name of the release definition in the upper right, I can see a status summary page of all releases that are still being retained within VSTS. This is where you can visually monitor the progress of every release in your pipeline for this definition, and a similar view is available by clicking All Release Definitions in the navigation tree on the left side of the portal (not shown).

Starting a Release

I now have a basic deployment pipeline between three environments: test, staging, and production, where the first two deployments happen automatically and the third is subject to manual approval. To start a release manually, I click + Release in the release definition and select Create Release. This prompts me with the dialog in **Figure 4** in which I select the build to use (from any successful build that's still in VSTS), and where I can also control the deployment chain between the environments. Notice how, for the Test environment, deployment happens automatically upon creation of the release, which is when I click the Create button. Staging and Production, similarly, have automatic deployment depending on the previous environment, subject, of course, to the success of any other release steps in those environments (such as tests), and any necessary approvals.

Once I've clicked Create and the release begins, I can navigate into that release to check on its progress, as shown in **Figure 5**.

I'll share that when I first created this release pipeline, using a Node.js back end, my build definition didn't create any actual artifacts, and so the release failed. I checked this by navigating to my most recent build and clicking the Artifacts tab on its summary page. Once I added the necessary Gulp task to produce some real output, as instructed by the guide on bit.ly/2c1XhTY, those artifacts were in place.

In my simple release pipeline, I haven't set up any automated tests so deployments to Test and Staging happen quickly in succession, and I can go to those Web sites and see the results. Before anything is deployed to Production, however, the release tells me that "A pre-deployment approval is pending for 'Production' environment. Approve or Reject," as you can

Continuous Deployment

Setting up continuous deployment means automatically starting a release after a successful build using the artifacts produced by that build. To do this, I edit the release definition and click on the Triggers tab, where there are options for Manual, Continuous Deployment and Scheduled releases. When I click on Continuous Deployment, I then select the source of the artifacts, which is my build definition for the back end, and save the release definition.

I can now go all the way out to Visual Studio, make a change in the back-end code, and commit it to the repository. This triggers a new build on VSTS because I checked the Continuous Integration box in the back end's build definition. Once that build succeeds, it automatically triggers a new release because I have Continuous Deployment checked in the release definition. Thus, with both CI and CD options, I've set up the full release pipeline between changes in the code and deployment to production, subject only to the manual pre-deployment approval that I specified for the Production environment in the release definition.

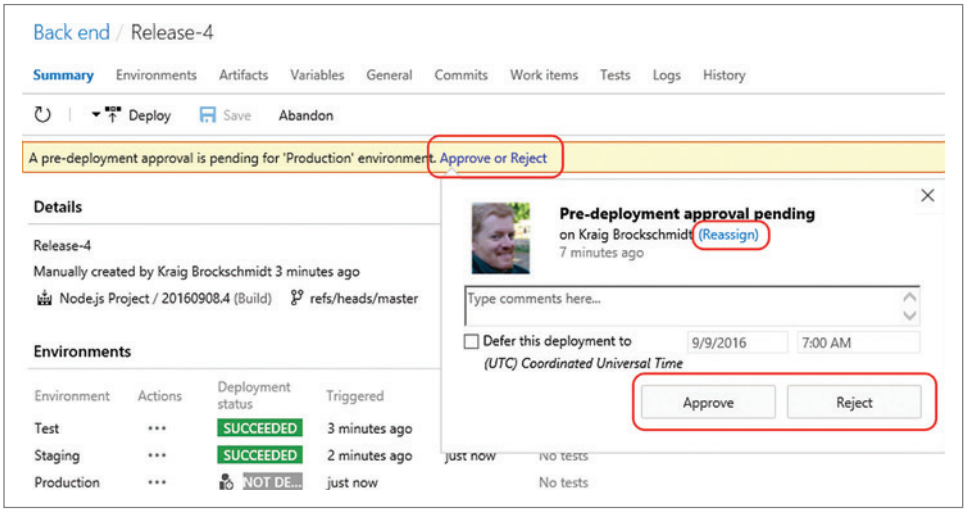


Figure 6 **Approving or Rejecting a Manual Approval Step**

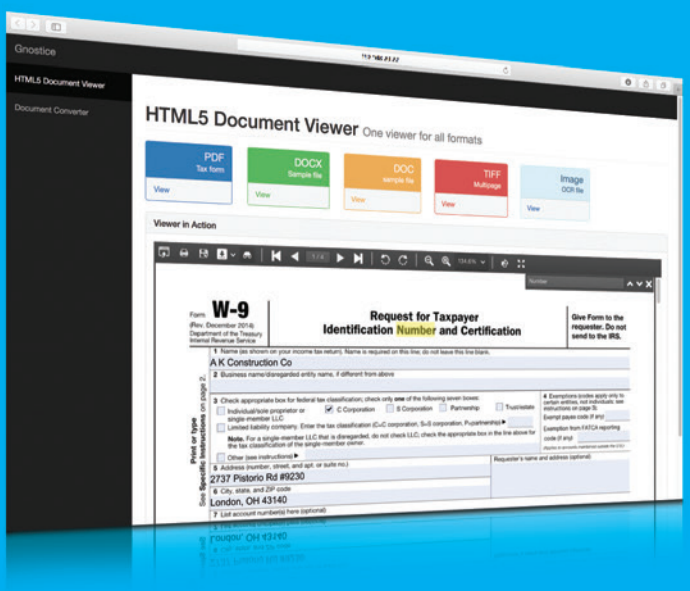
Document Technology for Everybody

PDF DOCX DOC RTF PPTX XLSX XLS TIFF MTIFF JPEG PNG BMP WMF EMF

HTML5 Web MVC WPF Win API API Java VCL FMX

Create | View | Print | Edit | Convert | Digitize

Get the Power of Great Design



Interactively Fill PDF Forms In Your Web App

- Responsive HTML5 control, automatically adjusts for Mobile, Desktop and Pad/Tablet.
- 100% Independent. No browser plug-ins required. No ActiveX.
- Single viewer for PDF, Office documents, text files and Images.
- Text Search, On-the-fly OCR on images, and more…
- Full-fledged client-side JavaScript to configure and perform all operations.
- TypeScript Support.

Royalty-Free Licensing

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

Download free trial from:
www.gnostice.com

Although this pipeline is simple, I now have a solid foundation on which I can build out additional steps, such as running any number of tests, simply by adding more tasks to the appropriate environment in the release definition, and setting up any necessary pre- and post-deployment approvals. I can also add new environments to divide the pipeline into even more distinct stages. But no matter how many tasks or environments you add, the basic process will be the same as you've seen here.

Walk-Through: Releasing the App

With the back-end release definitions in place, I can now create the three release definitions for the iOS, Android and Windows builds of the Xamarin app. The process is very much the same as with the back end, except that I start with a blank release template and select the platform-specific build definition. In the release definition editor, I won't have any tasks by default, so once I set up the environments I want (such as Test, Pre-Launch and Launch, to show you can use any names you want), I click on + Add task and select one from the dialog that appears.

In the case of the Android app, my deployments are as follows:

- **Test:** Use a build task to compile any test code, then use a Xamarin Test Cloud task to deploy the .apk file and test code to run those tests automatically on devices I've selected. (Of course, a Test Cloud account is necessary to use the service.)
- **Pre-Launch:** Use the HockeyApp task from the VSTS Marketplace, which I first install so it appears in the Add task dialog. I will have also created an account with HockeyApp and used its services to set up my list of pre-launch customers.
- **Launch:** For Android, I install the Google Play task from the Marketplace and select it from the Add task dialog (where you'll see Release, Promote and Increase Rollout tasks separately). This means, of course, that I've set myself up as a developer with Google.

For my iOS and Windows release definitions, I can still use Xamarin Test Cloud and HockeyApp, but those platforms don't provide for automated deployment to their respective stores. In these cases, my Launch environment doesn't have any tasks. Instead, I've simply assigned a pre-deployment approver to that environment. That person is responsible for evaluating feedback from the pre-launch testers and can then go out to the appropriate portals to upload the production app.

Pre-Launch Distribution with HockeyApp

HockeyApp (hockeyapp.net) is a powerful DevOps service for iOS, Android, and Windows apps both before and after a launch. Post-launch, HockeyApp provides crash reporting, usage data and user feedback, as will be covered in a later article. Pre-launch, HockeyApp provides these same services along with the ability to quickly and easily deploy test apps to any number of testers, no matter what devices they're using and no matter where they are in the world. It also lets you organize and manage your testers in a variety of ways so you can easily control which groups get which release, and when.

HockeyApp lets you organize and manage your testers in a variety of ways so you can easily control which groups get which release, and when.

Pre-launch distribution (bit.ly/2cxruZ8) works through the HockeyApp client that testers install on their devices. When you have a new test release ready, you upload it to the HockeyApp portal either manually or through a deployment step in VSTS. Your testers then receive an e-mail saying that the new release is available, and the HockeyApp client shows available releases that can be installed using the side-loading capabilities of the various mobile platforms.

Looking Ahead

In this series of articles, I've now looked at source control, build, and release management, through which you can create a full release pipeline across any number of environments that will accommodate whatever additional testing and approval steps you might require. This completes all the core connections between your source code and your customers. What remains, now, is to understand how to listen to those customers through monitoring of both apps and back-end services, and then to learn more about the variety of testing options you can employ, including Xamarin Test Cloud. ■

CodePush

Continuous deployment is most commonly used with Web applications and services because the deployment process is simply a matter of uploading new artifacts to the appropriate server. For mobile applications, automatic deployment is possible only for apps published to the Google Play store, but not at present for iOS and Windows because both require some manual steps. Also, approval of app updates can take as long as two weeks, which makes it difficult to get even simple, much-less-critical bug fixes out to customers. Fortunately, apps built using Apache Cordova and React Native can take advantage of the Microsoft CodePush service (in preview at the time of writing) to shortcut the process. CodePush enables developers to automatically deploy HTML, CSS, JavaScript and static artifacts like images directly to customer devices. The service works by having the app query CodePush for updates that are then applied to the running app, thereby removing the need to go through the app store approval process. (This practice is allowed by app store policies, provided the app's original purpose doesn't change.) Learn more at microsoft.github.io/code-push.

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

Thanks to the following technical expert for reviewing this article: *Alex Homer*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

7 LOCATIONS TO CHOOSE FROM

JOIN US

ROCK YOUR CODE TOUR 2017

LAS VEGAS
MARCH 13-17

See pages 68-71

AUSTIN
MAY 15-18

See pages 42-43

CHICAGO
SEPT 18-21

WASH. DC
JUNE 12-15

See pages 44-45

ANAHEIM
OCT 16-19

REDMOND
AUG 14-18

ORLANDO
NOV 13-17

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com

TURN THE PAGE FOR MORE EVENT DETAILS →

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

AUSTIN, TX
MAY 15-18, 2017
HYATT REGENCY

EVENT PARTNER
Microsoft

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

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 March 17 and Save $300!

Use promo code VSLJAN2

REGISTER NOW

ROCK YOUR CODE TOUR 2017

AUSTIN
MAY 15-18

"I felt more capable of creating Web Applications as a result of this conference. It was empowering and liberating." – David Pfahler, Dell, Inc.

"The sessions were packed full of extremely valuable info. Austin is a killer location for conference!!" – Aaron Eversole, Software Solutions Integrated, LLC

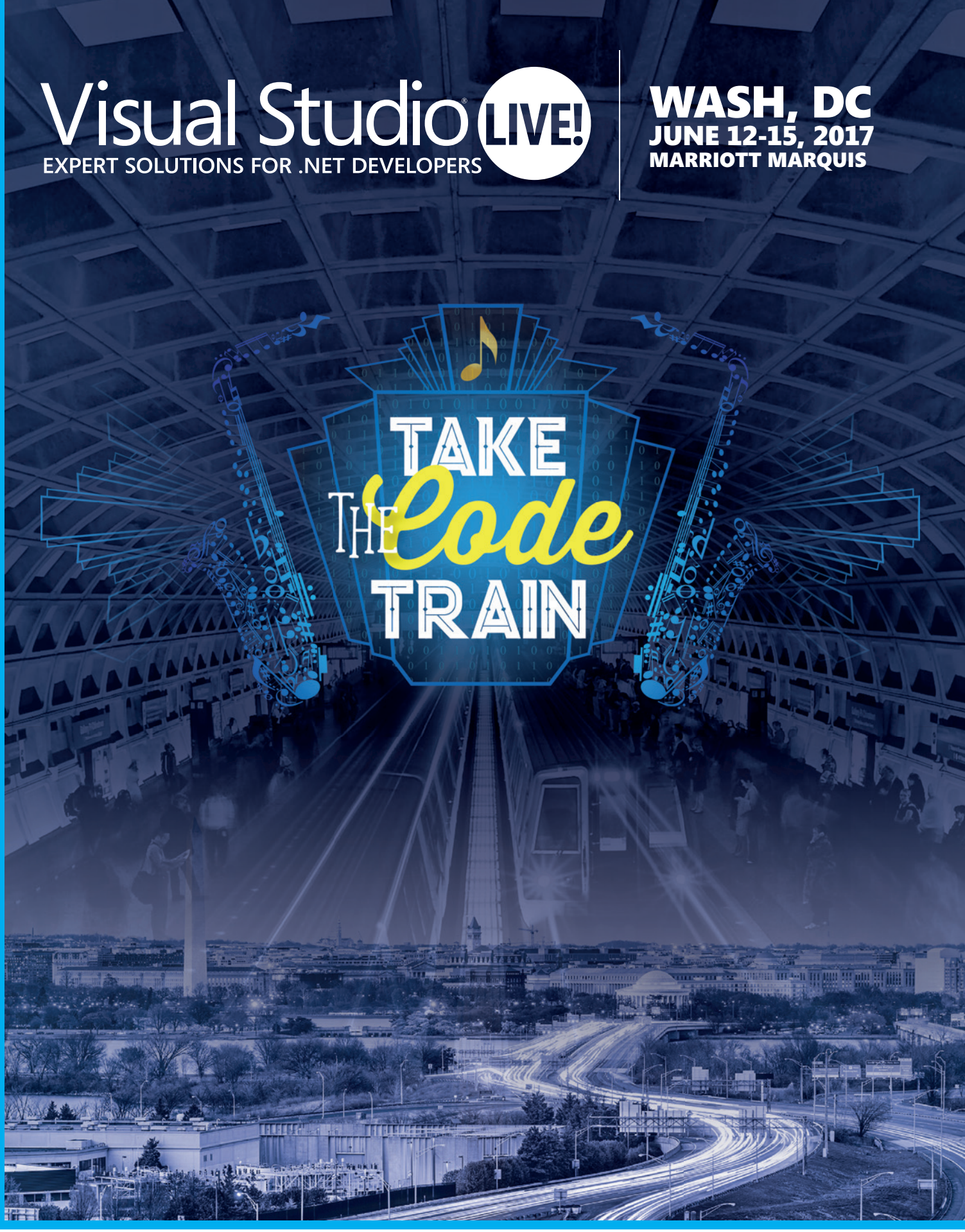
vslive.com/austinmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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



EVENT PARTNER

Microsoft

SUPPORTED BY

Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

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 April 21 and Save $300!

Use promo code VSLJAN2

REGISTER NOW

ROCK YOUR CODE TOUR 2017

WASH. DC

JUNE 12-15

"I like that this event is in D.C. so I can actually attend! I learned about features that we could leverage as well as information about trends we are seeing in the industry at the cutting edge." – Jonathan Cogan, Geico

"Very intelligent speakers. I really liked the topics covering new .NET and C# tools/technology and code." – Jon Bleichner, NIH

vslive.com/dcmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Introduction to the Microsoft CNTK Machine Learning Tool

James McCaffrey

The Microsoft Computational Network Toolkit (CNTK) is a very powerful command-line system that can create neural network prediction systems. Because CNTK was originally developed for internal use at Microsoft, the documentation is a bit intimidating. In this article I walk you through the process of installing CNTK, setting up a demo prediction problem, creating a neural network model, making a prediction,and interpreting the results.

Figure 1 gives you an idea of what the CNTK is and a preview of where this article is headed. Although it's not visible in the image, I ran the CNTK tool by entering the command:

```
> cntk.exe configFile=MakeModel.cntk makeMode=false
```

The image shows only the last part of the output messages. The configuration file with a .cntk extension contains information about input files and about the design of a neural network to use. A prediction model was created and saved to disk.

After creating the prediction model, I used it to make a prediction. The input values are in file NewData.txt and are (4.0, 7.0).

This article discusses:

- Installing Microsoft CNTK
- Setting up a demo prediction problem
- Creating a neural network model
- Making a prediction and interpreting the results

Technologies discussed:

Microsoft CNTK

Code download available at:

msdn.com/magazine/0117magcode

The CNTK tool was used with a second configuration file named MakePrediction.cntk to compute a prediction. The prediction results were saved to file Prediction_txt.pn and are (0.271139, 0.728821, 0.000040), which means that the predicted outcome is the second of three possible output values.

This article assumes you have familiarity working with command-line programs and a rough idea of what neural networks are, but doesn't assume you're a machine learning (ML) expert or know anything about CNTK. You can also get the code and data from the accompanying download.

Setting up the Problem

Imagine a scenario where you want to predict a person's political leaning (conservative, moderate, liberal) from their age and annual income. This is called a classification problem. The idea is to take some data with known values and create a prediction model. Here, you can think of the model as a kind of complex math function that accepts two numeric input values and then emits a value indicating one of three classes.

Now take a look at the graph in **Figure 2**. There are 24 data points. The two input variables, called features in ML terminology, are x0 and x1. The three colors indicate three different classes, sometimes called labels in ML terminology. Although a human being can quickly see a pattern, trying to create a prediction model for even this simple data set is very challenging for a computer system.

After CNTK uses the data shown in **Figure 2** to create a neural network prediction model, the model will be applied to the nine test data points shown in **Figure 3**. As you'll see, CNTK correctly predicts eight out of the nine test cases. The test item at (8.0, 8.0), which is actually class "red," will be incorrectly predicted to be class "blue."

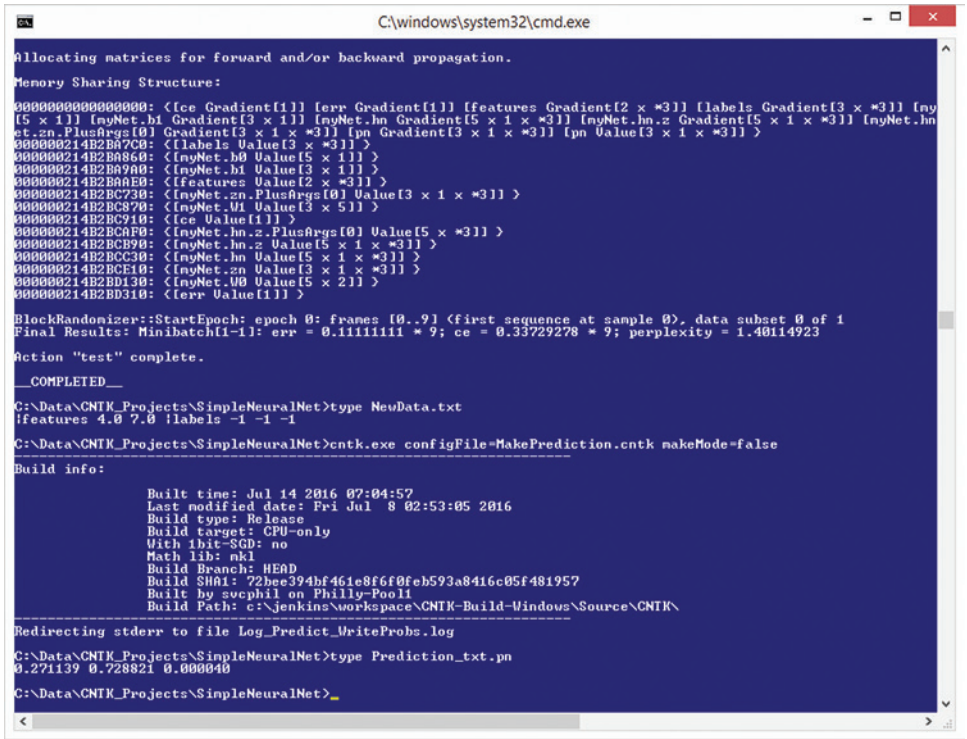


Figure 1 **CNTK in Action**

Installing CNTK

Installing CNTK is just a matter of downloading a .zip folder from GitHub and extracting the files. The main CNTK portal site is located at github.com/Microsoft/CNTK. On that page you'll find a link to the current releases (github.com/Microsoft/CNTK/releases). One of the keys to the power of CNTK is that it can optionally use a GPU instead of your machine's CPU. The releases page gives you the choice to download binaries for a CPU-only version or a GPU+CPU version.

For demonstration purposes, I recommend you select the CPU-only version, even though you can direct the GPU+CPU version to use only the CPU. Clicking on the associated link on the releases page brings you to a page where you'll need to accept some licensing terms, and after you click the Accept button, you'll get a dialog where you can save a file with a name something like CNTK-1-6-Windows-64bit-CPU-Only.zip (the version number could be different, of course) to your machine.

Download the .zip file to your desktop or any convenient directory. Then extract all files directly to the C: drive (most common) or to the C:\Program Files directory.

The extracted download will have a single root directory named cntk. That root directory will contain several directories, including another directory also named cntk that contains all the binaries, including the key cntk.exe file. To finish the installation process, add the path to the cntk.exe file to your system PATH environment variable (typically C:\cntk\cntk).

Creating the Data Files

To create and run a CNTK project, you need a configuration file with a .cntk extension, and at least one file that contains training data. Most CNTK projects will have a file of test data, too. Additionally, the CNTK tool will create several output files when a project is run.

There are many ways to organize the files used with a CNTK project. I recommend you create a project root directory that holds your data files and the .cntk configuration file, and run CNTK from that directory.

I had an existing C:\Data directory on my machine. For the demo, I created a new subdirectory in that directory named CNTK_Projects.

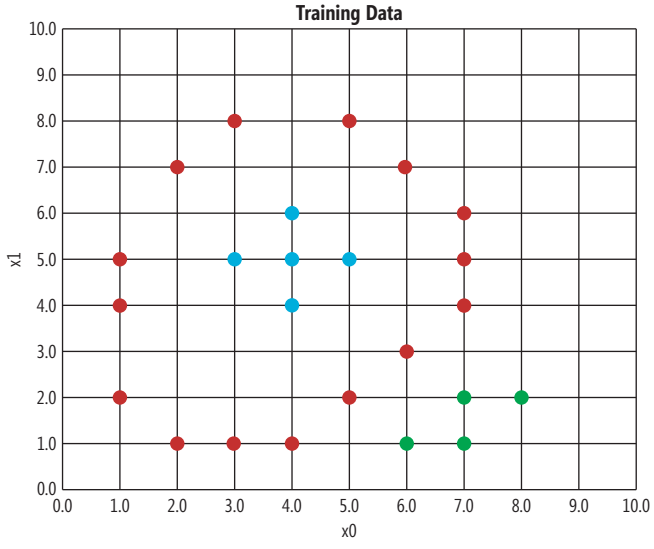


Figure 2 **Training Data**

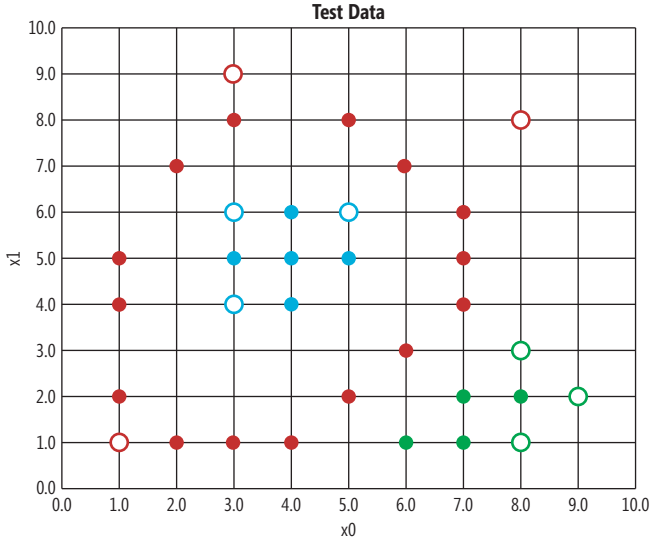


Figure 3 **Test Data (Open Circles)**

Figure 4 **The Training Data**

```
|features 1.0 5.0 |labels 1 0 0
|features 1.0 2.0 |labels 1 0 0
|features 3.0 8.0 |labels 1 0 0
|features 4.0 1.0 |labels 1 0 0
|features 5.0 8.0 |labels 1 0 0
|features 6.0 3.0 |labels 1 0 0
|features 7.0 5.0 |labels 1 0 0
|features 7.0 6.0 |labels 1 0 0
|features 1.0 4.0 |labels 1 0 0
|features 2.0 7.0 |labels 1 0 0
|features 2.0 1.0 |labels 1 0 0
|features 3.0 1.0 |labels 1 0 0
|features 5.0 2.0 |labels 1 0 0
|features 6.0 7.0 |labels 1 0 0
|features 7.0 4.0 |labels 1 0 0
|features 3.0 5.0 |labels 0 1 0
|features 4.0 4.0 |labels 0 1 0
|features 5.0 5.0 |labels 0 1 0
|features 4.0 6.0 |labels 0 1 0
|features 4.0 5.0 |labels 0 1 0
|features 6.0 1.0 |labels 0 0 1
|features 7.0 1.0 |labels 0 0 1
|features 8.0 2.0 |labels 0 0 1
|features 7.0 2.0 |labels 0 0 1
```

And inside that directory I created a subdirectory named SimpleNeuralNet to act as the demo project root directory to hold my .cntk file and a training data file and a test data file.

The CNTK system can work with several different types of data files. The demo uses simple text files. Open an instance of Notepad and use the 24 data items from **Figure 4**, or manually create the data using the information in **Figure 2**, then save the file as TrainData.txt in the SimpleNeuralNet directory.

The training data looks like:

```
|features 1.0 5.0 |labels 1 0 0
|features 1.0 2.0 |labels 1 0 0
. . .
|features 7.0 2.0 |labels 0 0 1
```

The "|features" tag indicates input values and the "|labels" tag indicates output values. Neither "features" nor "labels" are reserved words, so you could use something like "|predictors" and "|predicteds" if you prefer. Values can be delimited using a blank space or the tab character (the demo data uses blank spaces). Neural networks only understand numeric values so class labels like "red" and "blue" must be encoded as numbers. Neural network classifier models use what's called 1-of-N encoding. For three possible class labels, you'd use 1 0 0 for the first class ("red" in the demo), 0 1 0 for the second class ("blue") and 0 0 1 for the third class ("green"). It's up to you to keep track of how each label value is encoded.

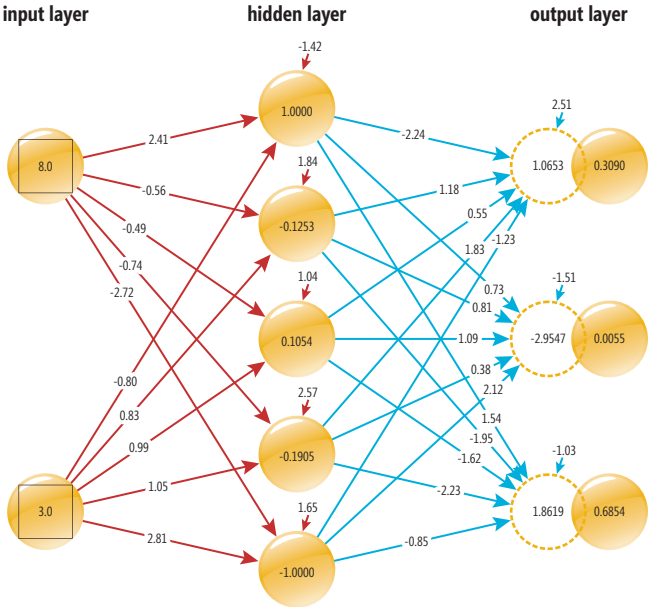


Figure 5 **Neural Network Input and Output**

In a non-demo scenario, you might have to worry about input data normalization. In situations where the input values differ greatly in magnitude, you get better results if you scale data so that all magnitudes are roughly the same. For example, suppose the input data is a person's age (like 32.0) and annual income (like $48,500.00). You could preprocess the data by dividing all age values by 10 and all income values by 10,000, giving normalized input values like (3.20, 4.85). The three most common forms of input data normalization are called z-score normalization, min-max normalization and order of magnitude normalization.

After you've created and saved the TrainData.txt file, create and save the following nine items of test data as TestData.txt file in the SimpleNeuralNet directory:

```
|features 1.0 1.0 |labels 1 0 0
|features 3.0 9.0 |labels 1 0 0
|features 8.0 8.0 |labels 1 0 0
|features 3.0 4.0 |labels 0 1 0
|features 5.0 6.0 |labels 0 1 0
|features 3.0 6.0 |labels 0 1 0
|features 8.0 3.0 |labels 0 0 1
|features 8.0 1.0 |labels 0 0 1
|features 9.0 2.0 |labels 0 0 1
```

Understanding Neural Network Input and Output

To understand how to use CNTK you need a basic understanding of what a neural network is, how it computes output values and how to interpret those output values. Take a look at **Figure 5**. The diagram shows the neural network that corresponds to the demo problem.

There are two input nodes that hold values (8.0, 3.0), and three output nodes with values (0.3090, 0.0055, 0.6854). The network also has five hidden processing nodes.

Each input node has lines connecting it to all hidden nodes. And each hidden node has lines connecting it to all output nodes. These lines represent numeric constants called weights. Nodes are identified using 0-based indexing, so the top-most nodes are [0]. So, the weight from input[0] to hidden[0] is 2.41 and the weight from hidden[4] to output[2] is -0.85.

Each hidden node and each output node have an additional arrow. These are called the bias values. The bias for hidden[0] is -1.42 and the bias for output[2] is -1.03.

The input-output calculations are best explained with an example. First, hidden node values are computed. The middle hidden node, hidden[2] has value 0.1054 and is calculated by summing the products of all connected inputs and their associated weights plus the bias value, and then taking the hyperbolic tangent (tanh) of that sum:

```
hidden[2] = tanh( (8.0)(-0.49) + (3.0)(0.99) + 1.04) )
          = tanh( -3.92 + 2.98 + 1.04 )
          = tanh( 0.1058 )
          = 0.1054
```

The tanh function is called the hidden layer activation function. Neural networks can use one of several different activation functions.

GROUPDOCS
Document Manipulation APIs

Manipulating Files?

View, annotate, compare, convert, assemble, sign and share over **50 types of documents on the web.**

GroupDocs.**Viewer**
GroupDocs.**Annotation**
GroupDocs.**Conversion**
GroupDocs.**Comparison**
GroupDocs.**Signature**
GroupDocs.**Assembly**
GroupDocs.**Metadata**
GroupDocs.**Search**

Get started now

.NET Libraries | Java Libraries | Cloud APIs | Cloud Apps

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@groupdocs.com

Visit us at www.groupdocs.com

In addition to tanh, the other two most common are logistic sigmoid (usually shortened to just "sigmoid") and rectified linear.

After all the hidden node values are calculated, the next step is calculating the output nodes. First, the sum of the products of connected hidden nodes and their associated weights plus the bias value is computed. For example, output[0] for this step is 1.0653, calculated as:

```
output[0] = (1.0000)(-2.24) + (-0.1253)(1.18) +
            (0.1054)(0.55) + (-0.1905)(1.83) +
            (-1.0000)(-1.23) + 2.51

          = (-2.2400) + (-0.1478) +
            (0.0580) + (-0.3486) +
            (1.2300) + 2.51

          = 1.0653
```

In the same way, output[1] is calculated to be -2.9547 and output[2] is 1.8619.

Next, the three preliminary output values are scaled so they sum to 1.0, using what's called the softmax function:

```
output[0] = e^1.0653 / (e^1.0653 + e^-2.9547 + e^1.8619)
          = 0.3090

output[1] = e^-2.9547 / (e^1.0653 + e^-2.9547 + e^1.8619)
          = 0.0055

output[2] = e^1.8619 / (e^1.0653 + e^-2.9547 + e^1.8619)
          = 0.6854
```

These three values are interpreted as probabilities. So, for inputs of (8.0, 3.0) and the given weights and bias values, the outputs are (0.3090, 0.0055, 0.6854). The highest probability is the third value so the prediction is the third class, "green," in this case.

Another way of interpreting the output values is to map them so the highest probability is one and all others are zero. For this example you'd get (0, 0, 1), which maps to the encoded value of "green."

The process of determining the values of the weights and the biases is called training the network, and that's what CNTK does.

Creating the Configuration File

Figure 1 gives you an idea of how CNTK is used. In an ordinary command shell, I navigated to the project root directory at C:\Data\SimpleNeuralNet. The project root directory contains files TrainData.txt and TestData.txt and a MakeModel.cntk configuration file. The CNTK tool was invoked by executing the command:

```
> cntk.exe configFile=MakeModel.cntk makeMode=false
```

Recall that the system PATH variable knows the location of the cntk.exe program, so it doesn't have to be fully qualified. The .cntk configuration file has a lot of information. The makeMode=false parameter means to run the program and overwrite any previous results. CNTK command-line arguments are not case-sensitive.

Figure 6 shows the overall structure of the configuration file. The complete listing for the configuration file is presented in **Figure 7.**

You can name a CNTK configuration file however you wish, but using a .cntk extension is standard practice. You can use the # character or // token for comments, which don't span lines. At the top of the configuration file you give a colon-delimited list of modules to run, four in this case:

```
command=Train:WriteProbs:DumpWeights:Test
```

Notice that the order in which modules are executed doesn't have to match the order in which they're defined. Module names (in this example: Train, WriteProbs, DumpWeights, Test) aren't keywords so you can name modules as you wish. Notice the Train module has an instruction action="train." Here, both words are keywords.

The Train module uses a training data file to create the prediction model, so most CNTK configuration files will have a module, usually named Train, that contains an action="train" command. The Train module will write the resulting model information to disk in a binary format.

The Test module is optional but is usually included when creating a model. The Test module will use the newly created prediction model to evaluate the overall prediction accuracy and prediction error on the training data.

The WriteProbs module is optional. The module will write the actual prediction values for the test data items to a text file in the project root directory. This allows you to see exactly which test cases were correctly predicted and which were not.

The DumpWeights module will write a text file that contains the neural network weights and biases that define the prediction model. You can use this information to uncover trouble spots, and to make predictions on new, previously unseen data.

System Parameters

The MakeModel.cntk configuration file sets up five system parameters:

```
modelPath = "Model\SimpleNet.snn"
deviceId = -1
dimension = 2
labelDimension = 3
precision = "float"
```

The modelPath variable specifies where to put the resulting binary model and what to call the model. Here, "snn" stands for simple neural network but you can use any extension. The deviceId variable tells CNTK whether to use the CPU (-1) or the GPU (0).

The dimension variable specifies the number of values in an input vector. The labelDimension specifies the number of possible output values. The precision variable can take values of float or double. In most cases float is preferable because it makes training much faster than double.

The Training Module

The demo Train module in the configuration file has three major sub-sections: BrainScriptNetworkBuilder, SGD and reader. These sub-sections define the neural network architecture, how to train the network and how to read training data.

Figure 6 **The Structure of the Configuration File**

```
# MakeModel.cntk

command=Train:WriteProbs:DumpWeights:Test
# system parameters go here

Train = [
  action="train"
  BrainScriptNetworkBuilder = [
    # define network here
  ]
]
Test = [
  # training commands here
]
WriteProbs = [
  # output commands here
]
DumpWeights = [
  # output commands here
]
```

The training module definition begins as:

```
Train = [
  action="train"

  BrainScriptNetworkBuilder = [

    FDim = $dimension$
    HDim = 5
    LDim = $labelDimension$
...
```

The BrainScriptNetworkBuilder section of the Train module uses a special scripting language called BrainScript. Variables FDim, HDim, and LDim hold the number of features, hidden nodes, and label nodes for the neural network. These names aren't required, so you could use names like NumInput, NumHidden and NumOutput if you wish. The number of input and output nodes is determined by the problem data, but the number of hidden nodes is a free parameter and must be determined by trial and error. The $ token is a substitution operator. The Train module definition continues with:

```
  neuralDef (ftrs) = [
    W0 = Parameter (HDim, FDim)
    b0 = Parameter (HDim, 1)
    W1 = Parameter (LDim, HDim)
    b1 = Parameter (LDim, 1)

    hn = Tanh (W0 * ftrs + b0)
    zn = W1 * hn + b1
  ].zn
  ...
```

Figure 7 **The Training Configuration File**

```
# MakeModel.cntk

command=Train:WriteProbs:DumpWeights:Test

modelPath = "Model\SimpleNet.snn"
deviceId = -1
dimension = 2
labelDimension = 3
precision = "float"

# =====
Train = [
  action="train"

  # network description
  BrainScriptNetworkBuilder = [

    FDim = $dimension$
    HDim = 5
    LDim = $labelDimension$

    # define the neural network
    neuralDef (ftrs) = [
      W0 = Parameter (HDim, FDim)
      b0 = Parameter (HDim, 1)
      W1 = Parameter (LDim, HDim)
      b1 = Parameter (LDim, 1)

      hn = Tanh (W0 * ftrs + b0)
      zn = W1 * hn + b1
    ].zn

    # specify inputs
    features = Input (FDim)
    labels   = Input (LDim)

    # create network
    myNet = neuralDef (features)

    # define training criteria and output(s)
    ce   = CrossEntropyWithSoftmax (labels, myNet)
    err  = ErrorPrediction (labels, myNet)
    pn   = Softmax (myNet)

    # connect to the NDL system.
    featureNodes    = (features)
    inputNodes      = (labels)
    criterionNodes  = (ce)
    evaluationNodes = (err)
    outputNodes     = (pn)
  ]

  # stochastic gradient descent
  SGD = [
    epochSize = 0
    minibatchSize = 1
    learningRatesPerSample = 0.04
    maxEpochs = 500
    momentumPerMB = 0.90
  ]

  # configuration for reading data
  reader = [
    readerType = "CNTKTextFormatReader"
    file = "TrainData.txt"
    input = [
      features = [
        dim = $dimension$
        format = "dense"
      ]
      labels = [
        dim = $labelDimension$
        format = "dense"
      ]
    ]
  ]
]

# test
Test = [
  action = "test"
  reader = [
    readerType = "CNTKTextFormatReader"
    file="TestData.txt"
    randomize = "false"
    input = [
      features = [
        dim = $dimension$
        format = "dense"
      ]
      labels = [
        dim = $labelDimension$
        format = "dense"
      ]
    ]
  ]
]

# log the output node values
WriteProbs = [
  action="write"
  reader=[
    readerType="CNTKTextFormatReader"
    file="TestData.txt"
    input = [
      features = [
        dim = $dimension$
        format = "dense"
      ]
      labels = [
        dim = $labelDimension$
        format = "dense"
      ]
    ]
  ]
  outputPath = "TestProbs_txt"
]

# dump weight and bias values
DumpWeights = [
  action = "dumpNode"
  printValues = "true"
]
```

This is a BrainScript function. Variable W0 is a matrix that holds the input-to-hidden weights. The Parameter function means "construct a matrix." Variable b0 holds the hidden node bias values. All calculations in BrainScript are performed on matrices, so b0 is a matrix with one column rather than an array.

Variables W1 and b1 hold the hidden-to-output weights and the output node bias values. The values of the hidden nodes are calculated into a variable named hn using a sum of products and the tanh function, as explained earlier. Variable zn holds the pre-softmax output values. The closing bracket-dot-variable notation is how a BrainScript function returns a value. The Train definition continues with:

```
features = Input (FDim)
labels   = Input (LDim)
myNet = neuralDef (features)
...
```

Here, the input features and output labels are defined. Variable names "features" and "labels" aren't keywords, but they must match the strings used in the training and test data files. The neural network is created by calling the neuralDef function. Next, the module defines information that will be used during training:

```
ce   = CrossEntropyWithSoftmax (labels, myNet)
err  = ErrorPrediction (labels, myNet)
pn   = Softmax (myNet)
...
```

The CrossEntropyWithSoftmax function specifies that cross-entropy error should be used when calculating how close calculated output values are to actual output values in the training data. Cross-entropy error is the standard metric but squared error is an alternative.

The ErrorPrediction function instructs CNTK to compute and display the prediction model accuracy (percentage of correct predictions on the training data) and cross-entropy error and perplexity, which are measures of error between calculated outputs and actual outputs.

The Softmax function instructs CNTK to normalize computed output values so they sum to 1.0 and can be interpreted as probabilities. For a neural network classifier, Softmax is used except in extremely rare situations. The training module definition concludes with:

```
...
  featureNodes    = (features)
  inputNodes      = (labels)
  criterionNodes  = (ce)
  evaluationNodes = (err)
  outputNodes     = (pn)
]
```

Here, the required system variables of featureNodes, inputNodes, criterionNodes, and outputNodes, and the optional evaluationNodes, are associated with the user-defined variables.

The stochastic gradient descent (SGD) sub-section defines how CNTK will train the neural network. In the context of a neural network, SGD is usually called back-propagation. The sub-section definition is:

```
SGD = [
  epochSize = 0
  minibatchSize = 1
  learningRatesPerSample = 0.04
  maxEpochs = 500
  momentumPerMB = 0.90
]
```

The epochSize variable specifies how much of the training data to use. A special value of zero means to use all available training data. The minibatchSize variable specifies how much of the training data to process in each training iteration. A value of one means update the weights and biases after each training item is processed. This is often called "online" or "stochastic" training.

If the value of minibatchSize is set to the number of training items (24 in the case of the demo), then all 24 items would be processed and the results aggregated, and only then would the weights and biases be updated. This is sometimes called "full-batch" or "batch" training. Using a value between one and the training set size is called "mini-batch" training.

The Softmax function instructs CNTK to normalize computed output values so they sum to 1.0 and can be interpreted as probabilities.

The learningRatesPerSample variable specifies how much to adjust the weights and biases on each iteration. The value of the learning rate, along with the other parameters in the SGD sub-section, must be determined by trial and error. Neural networks are typically extremely sensitive to the value of the learning rate—for example, using 0.04 might give you a very accurate prediction system but using 0.039 or 0.041 could give you a very poor system.

The maxEpochs variable specifies how many iterations to perform for training. Too small a value will result in a poor model ("model under-fit"), but too many iterations will over-fit the training data. This leads to a model that predicts the training data very well but predicts new data very poorly.

The momentumPerMB (momentum per mini-batch, not per megabyte as you might assume) is a factor that increases or decreases the amount by which weights and biases are updated. Just like the learning rate, a momentum value must be determined by trial and error, and neural network training is typically extremely sensitive to the value of momentum. The value of 0.90 used by the demo is the default value so the momentumPerMB parameter could have been omitted.

The training module of the demo configuration file concludes by setting the values of the parameters of the reader sub-section:

```
...
  reader = [
    readerType = "CNTKTextFormatReader"
    file = "TrainData.txt"
    input = [
      features = [ dim = $dimension$; format = "dense" ]
      labels = [ dim = $labelDimension$; format = "dense" ]
    ]
  ]
] # end Train
```

The CNTK tool has many different types of readers for use with different types of input data. The meaning of the parameters in the demo reader sub-section should be clear. Notice that I put multiple statements on a single line by using the semicolon delimiter.

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 $400
USE PROMO CODE TMJAN1

[TECHMENTOREVENTS.COM/REDMOND]

SUPPORTED BY: Redmond | Redmond Channel Partner | Virtualization Review

PRODUCED BY: 1105 MEDIA

The Test Module

To recap to this point, the MakeModel.cntk configuration file has some global system parameters (such as modelPath) plus four modules: Train, Test, WriteProbs, DumpWeights. The Train module has three sub-sections: BrainScriptNetworkBuilder, SGD, reader.

The Test module is mercifully very simple, as you can see in **Figure 8**.

The reader sub-section of the test module should match the reader sub-section of the training module except for the file parameter value and the addition of the randomize parameter. When training with SGD, it's extremely important that the data items be processed in random order, and true is the default value for randomize. But when walking through the test data there's no need to randomize the order of the data items.

The test module emits one accuracy metric and two error metrics to the shell. If you refer back to **Figure 1**, just before the "Action test complete" message, you'll see:

```
err = 0.11111111 * 9
ce = 0.33729280 * 9
perplexity = 1.40114927
```

The err = 0.1111 * 9 means that 11 percent of the nine test data items were incorrectly predicted using the model. In other words, eight out of nine test items were correctly predicted. The training output doesn't, however, tell you which data items were correctly and incorrectly predicted.

The ce = 0.3372 * 9 means that the average cross-entropy error is 0.3372. For this introduction to CNTK, just think of cross entropy as an error term, so smaller values are better.

The perplexity = 1.4011 is a minor metric. You can think of perplexity as a measure of how strong the predictions are, where smaller values are better. For example, for three possible output values as in the demo, if the prediction is (0.33, 0.33, 0.33) ,you don't have a strong prediction at all. The perplexity in this case would be 3.0, which is a maximum for three output values.

The WriteProbs Module

The third module in the demo CNTK configuration file is WriteProbs. This module is optional but very useful because it gives you additional information about the predictions made on the test data. The module is defined in **Figure 9**.

The WriteProbs module is the same as the test module except for three changes. First, the action parameter is set to "write" instead of "test." Second, the randomize parameter has been removed (because false is the default). Third, an outputPath parameter has been added.

When the WriteProbs module executes, it will write the exact output values for the test data to the specified file. In this case the file name will have ".pn" appended because that was the variable name used for the output nodes in the training module.

For the nine demo test items, the contents of file TestProbs_txt.pn are:

```
0.837386 0.162606 0.000008
0.990331 0.009669 0.000000
0.275697 0.724260 0.000042

0.271172 0.728788 0.000040
0.264680 0.735279 0.000041
0.427661 0.572313 0.000026

0.309024 0.005548 0.685428
0.000134 0.000006 0.999860
0.000190 0.000008 0.999801
```

The first three probability vectors go with the first three test items, which map to the correct output of (1, 0, 0), so the first two test items were predicted correctly. But the third probability vector of (0.27, 0.74, 0.00) maps to (0, 1, 0), so it's an incorrect prediction.

The next three probability vectors go with test items that have output (0, 1, 0), so all three predictions are correct. Similarly, the last three probability vectors go with (0, 0, 1), so they're also correct predictions.

To recap, the Test module will emit accuracy and error metrics to the shell, but not tell you which individual test items are correct or give you their error. The WriteProbs module writes exact output values to file and you can use them to determine which test items are incorrectly predicted.

The DumpWeights Module

The last of the four modules in the demo configuration file is DumpWeights, which is defined as:

```
DumpWeights = [
  action = "dumpNode"
  printValues = "true"
]
```

When executed, this module will save the trained model's weights and bias values to file. By default, the filename will be the same as the binary model (SimpleNet.snn in the demo), with ".__AllNodes__.txt" appended, and the file will be saved in the directory specified by the modelPath parameter ("Model" in the demo).

After running the MakeModel.cntk demo, if you open a file explorer and point it to directory \SimpleNeuralNet\Model, you'll see 503 files:

```
SimpleNet.snn
SimpleNet.snn.__AllNodes__.txt
SimpleNet.snn.0
...
SimpleNet.snn.499
SimpleNet.ckp
```

Figure 8 **The Test Module**

```
Test = [
  action = "test"
  reader = [
    readerType="CNTKTextFormatReader"
    file = "TestData.txt"
    randomize = "false"
    input = [
      features = [ dim = $dimension$
        format = "dense" ]
      labels = [ dim = $labelDimension$
        format = "dense" ]
    ]
  ]
]
```

Figure 9 **The WriteProbs Module**

```
WriteProbs = [
  action="write"
  reader=[
    readerType="CNTKTextFormatReader"
    file="TestData.txt"
    input = [
      features = [  dim = $dimension$
        format = "dense" ]
      labels = [ dim = $labelDimension$
        format = "dense"  ]
    ]
  ]
  outputPath = "TestProbs_txt"
]
```

The SimpleNet.snn is the trained model saved in binary format for use by CNTK. The 500 files that have names that end with a digit, and the one file that ends with a ".ckp" extension, are binary checkpoint files. The idea here is that training a complex neural network can take hours or even days. Recall that the demo set a maxEpochs parameter to 500. The CNTK tool saves training information periodically so in case of a system failure, you don't have to restart training from scratch.

The first half of the contents of the AllNodes__.txt file for the demo (with a few lines removed) is:

```
myNet.b0=LearnableParameter [5,1]
-1.42185283
 1.84464693
 1.04422486
 2.57946277
 1.65035748
 ###############################################
myNet.b1=LearnableParameter [3,1]
 2.51937032
-1.5136646
-1.03768802
```

These are the values of the hidden node biases (b0) and the output node biases (b1). If you refer back to the neural network diagram in **Figure 4**, you'll see where these values are truncated to two decimals. The second half of the AllNodes__.txt file looks like:

```
myNet.W0=LearnableParameter [5,2]
 2.41520381 -0.806418538
-0.561291218 0.839902222
-0.490522146 0.995252371
-0.740959883 1.05180109
-2.72802472 2.81985259
 ###############################################
myNet.W1=LearnableParameter [3,5]
-2.246624  1.186315  0.557211  1.837152 -1.232379
 0.739416  0.814771  1.095480  0.386835  2.120146
 1.549207 -1.959648 -1.627967 -2.235190 -0.850726
```

Figure 10 **Making a Prediction**

```
# MakePrediction.cntk

stderr = "Log"   # write all messages to file
command=Predict:WriteProbs
modelPath = "Model\SimpleNet.snn" # where to find model
deviceId = -1
dimension = 2
labelDimension = 3
precision = "float"

Predict = [
  action = "eval"
  reader = [
    readerType="CNTKTextFormatReader"
    file="NewData.txt"
    input = [
      features = [ dim = $dimension$; format = "dense" ]
      labels = [ dim = $labelDimension$; format = "dense" ]
    ]
  ]
]

WriteProbs = [
  action="write"
  reader=[
    readerType="CNTKTextFormatReader"
    file="NewData.txt"
    input = [
      features = [ dim = $dimension$; format = "dense" ]
      labels = [ dim = $labelDimension$; format = "dense" ]
    ]
  ]
  outputPath = "Prediction_txt"  # dump with .pn extension
]
```

Recall that the demo network has two input values, five hidden nodes and three output nodes. Therefore, there are 2 * 5 = 10 input-to-hidden weights in W0, and there are 5 * 3 = 15 hidden-to-output weights in W1.

Making a Prediction

Once you have a trained model, you can use it to make a prediction. One way to do this is to use the CNTK tool with an "eval" action module. The demo takes this approach. First, a new set of data with a single item is created and saved as file NewData.txt:

```
|features 4.0 7.0 |labels -1 -1 -1
```

Because this is new data, the output labels use dummy -1 values. Next, I created a configuration file named MakePrediction.cntk with two modules named Predict and WriteProbs. The complete file is presented in **Figure 10**.

When run, the output probabilities are saved in a file named Prediction_txt.pn, which contains:

```
0.271139 0.728821 0.000040
```

This maps to output (0, 1, 0), which is "blue." If you look at the training data in **Figure 2**, you can see that (4.0, 7.0) could easily be either "red" (1, 0, 0) or "blue" (0, 1, 0).

Two alternative techniques for using a trained model are to use a C# program with the CNTK model evaluation library, or to use a custom Python script that uses the model weights and bias values directly.

> Once you have a trained model, you can use it to make a prediction.

Wrapping Up

To the best of my knowledge, CNTK is the most powerful neural network system for Windows that is generally available to developers. This article has covered only a very small part of what CNTK can do, but it should be enough to get you up and running with simple neural networks and allow you to understand the documentation. The real power of CNTK comes from working with deep neural networks—networks that have two or more hidden layers and possibly complicated connections between nodes.

The CNTK tool is under active development, so some of the details may have changed by the time you read this article. However, the CNTK team tells me that changes will likely be minor and you should be able to modify the demo presented in this article without too much difficulty. ■

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

Thanks to the following Microsoft technical experts who reviewed this article: *Adam Eversole, John Krumm, Frank Seide and Adam Shirey*

TED NEWARD

How To Be MEAN: Type Script with TypeScript

Welcome back, MEANers.

As this series on MEAN progressed over the last year and a half, one of the interesting changes to the topic list was to the "A" part of the series' name: AngularJS made a major switch by formally releasing version 2 of its line, and with that came some deep (and breaking) changes to the framework. One of the most striking changes was the choice to adopt TypeScript as the AngularJS "language of choice" for building AngularJS 2 applications, as opposed to plain, vanilla ECMAScript for the 1.x line.

Many readers are familiar with TypeScript, of course, because it's the latest language to emerge from the code mines in Redmond. For those not familiar with TypeScript, rest easy—TypeScript is actually similar to the ECMAScript 2015 syntax that you've read about earlier in this series, with a bit more by way of type information. Readers wanting a full dive into the language are encouraged to read Peter Vogel's article, "Understanding TypeScript," in the January 2015 issue of *MSDN Magazine* (msdn.com/magazine/dn890374).

However, AngularJS 2 uses a particular subset of features of TypeScript and, furthermore, not all readers are entirely comfortable with the TypeScript syntax yet. In addition, TypeScript has gone through a few changes since Vogel's article was written (version 2 dropped in late September 2015). Therefore, I want to take a quick pass through the language to make sure we're on the same page before addressing AngularJS 2.

So, let's talk some TypeScript.

Adding "Type" to "Script"

Conceptually, TypeScript is a straightforward idea: Take the traditional ECMAScript syntax and add some (optional) type information in the form of type annotations, similar to how F# and other functional languages provide type declarations. The TypeScript compiler (technically called a "transpiler" because it goes source-to-source, producing ECMAScript code out of the process) verifies that all type information is respected and obeyed; but the result is still good old, dynamically typed, browser-friendly JavaScript. In other words, the goal here is to obtain all the benefits of a type-safe language such as C# (reduction of obvious code errors through static verification) without either having to change the underlying JavaScript browser platform (good luck with that!) or build an expensive platform-on-top-of-another-platform. Because one of the core tenets of TypeScript is that "any legal ECMAScript program is also a legal TypeScript program," adopting TypeScript can be an incremental affair. Take small baby steps, getting cozy with new features only as much as the team feels comfortable in doing so, as opposed to jumping in entirely with new syntax (such as what one might need to do with an entirely new transpiled language such as CoffeeScript, Fantom or ClojureScript).

It's in that latter spirit that I begin this jaunt into TypeScript. I'll focus on the features that AngularJS 2 uses the most or most obvious and leave the rest for further exploration down the road.

Installing TypeScript

The first thing to note is that like most Node.js-based packages, TypeScript is an npm package. Thus, you install TypeScript via the usual "npm install" command:

```
npm install -g typescript
```

Because TypeScript will install a global command ("tsc"), it's important to use "-g," the "global" flag, when installing TypeScript. Take a moment and make sure you've fully installed the command-line tool by running "tsc":

```
$ tsc --version
Version 2.0.3
```

Therefore, with TypeScript installed, the next step is to write some TypeScript code.

Modules

The starting point for the discussion is that of TypeScript modules.

Presume that I create a file, person.ts, that's to contain a component. (The term "component" isn't one that TypeScript emphasizes, but AngularJS 2 does.) The first step is to create a simple function that can be invoked from another file, so let's first create that function:

```
function sayHello(message: string) {
  console.log("Person component says", message);
}
```

Notice the type annotation to the parameter, ensuring that the single parameter must be a string; this is the bedrock of TypeScript, and ensures that only strings can be passed as parameters. Granted, this function by itself makes a simple component, but complex or simple, it needs to be used to be useful.

So let's use it: a TypeScript application can use a component by using an import statement, like so:

```
import { sayHello } from './person';

sayHello("Fred");
```

In essence, the "import" statement declares that you're using the element named "sayHello" from the "person" module. (There are other forms of import syntax that you'll see later.) So, you run the two files, the aforementioned person.ts and this code app.ts, through the tsc compiler:

```
tsc person.ts app.ts
```

Unfortunately, TypeScript will complain, stating that person.ts is not a module.

Modules are, in TypeScript lingo, the language construct that provides a "box" around a tightly grouped set of code. As written, person.ts would be easily usable as a module under older JavaScript scenarios; simply defining a function in a file and referencing that file puts the function into the global scope. However, TypeScript requires more explicit syntax—you have to use the export keyword to declare what's part of the external surface area of the component, like so:

```
export function sayHello(message: string) {
  console.log("Person component says", message);
}
```

In TypeScript, any file that has a top-level import or export statement is considered a module, so simply declaring that this function is to be exported implicitly defines all of person.ts to be a module.

Figure 1 **A Simple Person Class**

```
export class Person {
  firstName: string;
  lastName: string;

  constructor(fn: string, ln: string) {
    this.firstName = fn;
    this.lastName = ln;
  }

  greet() : string {
    return this.fullName + " says hello!";
  }

  get fullName() : string {
    return this.firstName + " " + this.lastName;
  }
}
```

Figure 2 **Creating a Person**

```
export function createPerson(
  firstName: string, lastName: string, occupation: string) : Person {
  if (occupation == "Programmer")
    return new Programmer(firstName, lastName);
  else if (occupation == "Manager")
    return new Manager(firstName, lastName);
  else
    return new NormalPerson(firstName, lastName);
}

export interface Person {
  firstName: string;
  lastName: string;
  greet() : string;
  fullName: string;
}
```

Figure 3 **A Person Implementation**

```
class NormalPerson implements Person {
  firstName: string;
  lastName: string;

  constructor(fn: string, ln: string) {
    this.firstName = fn;
    this.lastName = ln;
  }

  greet() : string {
    return this.fullName + " says hello!";
  }

  get fullName() : string {
    return this.firstName + " " + this.lastName;
  }
}
```

Once modified, TypeScript is happy, and two new files, person.js and app.js, rest on the filesystem, waiting to be used.

Adding Class

TypeScript, like the ECMAScript 2015 language on which it's based, understands the core concept of classes, so it makes sense to define Person as a class to be used, as shown in **Figure 1**.

This is all relatively easy to figure out, even for those who have never looked at TypeScript before. The export keyword again indicates that this class is for use outside of this module. The fields firstName and lastName use TypeScript annotations to get the compiler to enforce "string-ness," the method greet returns a string to callers, and the fullName method is declared as a synthetic read-only property accessor, made up of the firstName and lastName fields. Using the Person type in the app.ts file is also straightforward—you just need to import the Person type from the person.ts file and construct one using the keyword new:

```
import { Person } from './Person';

let ted = new Person("Ted", "Neward");
console.log(ted.greet());
```

Careful readers will note that the import line has changed—instead of pulling in sayHello, it pulls in the Person type. While it would certainly be possible to list all of the symbols exported in Person between the brackets of the import statement, that would get truly tedious very quickly. So TypeScript provides a wildcard import facility, but because you don't want all of the module's exported names to just pollute the global namespace, you need to provide a name under which all of those names will be visible. Using it would change the application code slightly:

```
import * as PerMod from './Person';

let ted = new PerMod.Person("Ted", "Neward");
console.log(ted.greet());
```

Obviously, this isn't production-quality code, because PerMod is a terrible name.

Interfacing in TypeScript

Of course, one popular goal for component-based development (which, remember, AngularJS 2 stresses) is that there should be a strong separation between how users of a component utilize the component and how the component provides that utility—in other words, the "interface vs. implementation" distinction. TypeScript takes a page from its conceptual sibling C# here, providing the ability to declare interfaces—which, like in C#, are promises of behavior that an implementation will provide.

So if the Person component wants to distinguish between different kinds of Persons without requiring any implementation restrictions, it can define Person as an interface, provide several different implementations, and maybe a constructor function to make it easy to construct Persons without having to worry about the details between them, as shown in **Figure 2**.

Creating a class that implements Person is straightforward, using the implements keyword, as shown in **Figure 3**.

And, as shown in **Figure 4**, creating subtypes of NormalPerson (for managers and programmers) is equally straightforward, creating a constructor that will defer to its parent class and then overriding the greet method to return messages appropriate to each occupation.

Again, aside from the type descriptors (and the interface declaration itself), this is similar to straight-up ECMAScript 2015 syntax, but thanks to TypeScript type-checking, any attempt to use anything other than a string as the parameters to the constructors will be firmly rejected. Note, however, that the fact the classes aren't exported means that the client code has no idea what the actual implementation is; all the client knows is that the Person interface defines three properties—firstName, lastName and fullName—and one method—greet—that the client can use.

Decorations

The last obvious feature of TypeScript that requires explanation is decorators, an experimental feature for ECMAScript (and TypeScript, for that matter) that look vaguely like custom attributes, but behave quite differently. Essentially, by using an @-prefixed notation, you can define a function that will be invoked whenever different code constructs are invoked—when a class is constructed, when a method is invoked, when properties are accessed (or modified), or even when parameters are passed as part of a method or function invocation. It's an obvious attempt to provide some aspect-oriented programming approaches to TypeScript, and AngularJS 2 leverages it pretty heavily.

The classic example for an AOP library is that of logging function calls; you'd like to reuse code that logs to console every time a particular function or method is called, regardless of from where that call comes.

Figure 4 **A Programmer Implementation**

```
class Programmer extends NormalPerson {
  constructor(fn: string, ln: string) {
    super(fn, ln);
  }

  greet() : string {
    return this.fullName + " says Hello, World!";
  }
}

class Manager extends NormalPerson {
  constructor(fn: string, ln: string) {
    super(fn, ln);
  }

  greet() : string {
    return this.fullName + " says let's dialogue about common synergies!";
  }
}
```

Figure 5 **Defining the Logging Annotation**

```
export default function log() {
  return function(target: any,
    propertyKey: string,
    descriptor: PropertyDescriptor)
  {
    // Save a reference to the original method
    var originalMethod = descriptor.value;
    descriptor.value = function (...args: any[]) {
      var argsLog = args.map(a => JSON.stringify(a)).join();
      var result = originalMethod.apply(this, args);
      var resultLog = JSON.stringify(result);
      console.log(`Call: ${propertyKey}(${argsLog}) => ${resultLog}`);
      return result;
    }

    // Return edited descriptor instead of overwriting
    // the descriptor by returning a new descriptor
    return descriptor;
  }
}
```

This, by definition, is a cross-cutting concern, a chunk of code that defies traditional object-oriented reuse constructs such as inheritance. Using TypeScript, you can write a log decorator, and apply that decorator to the methods that you want to decorate with the logging behavior. That behavior is invoked whenever the decorated method is invoked.

In practical terms, this means that if you've written a log decorator, the Person implementation returned can use @log on the greet method, and calls will be logged to console, as shown here:

```
import log from './log';

// ... Code as before

class Manager extends NormalPerson {
  constructor(fn: string, ln: string) {
    super(fn, ln);
  }

  @log()
  greet() : string {
    return this.fullName + " says let's dialogue about common synergies!";
  }
}
```

When run, it produces some nice method-level logging:

```
$ node app.js
Call: greet() => "Ted Neward says Hello, World!"
Ted Neward says Hello, World!
Call: greet() => "Andy Lientz says let's dialogue about common synergies!"
Andy Lientz says let's dialogue about common synergies!
Call: greet() => "Charlotte Neward says hello!"
Charlotte Neward says hello!
```

The log component itself is a nifty bit of runtime type-mongery, but a bit beyond the scope of this column. It's included in **Figure 5** for your perusal, but I won't describe how it works here other than to say that TypeScript will effectively inject some code into the right places to make calls to the function that the log decorator returns.

The TypeScript Web site describes decorators at bit.ly/2fh1lzC, for those who want to know how to create them. Fortunately, knowing how to create your own decorators isn't required to use AngularJS 2 effectively; however, knowing how to use decorators that already exist is definitely a requirement. For starters, AngularJS 2 uses them for dependency injection, a core staple of "The Angular Way" since its inception.

Wrapping Up

I've taken a quick pass through TypeScript, and while it's definitely not an exhaustive treatment, it'll get you going once you pick up AngularJS 2 and start hacking on that. Note that some of the features described here will require particular compiler switches; decorators, in particular, will require the compiler switch —experimentalDecorators (or its equivalent in tsconfig.json). Most of the time, however, the Yeoman-generated scaffolding will have the right switches already in place, and AngularJS 2 developers don't need to worry about them.

Speaking of which, it's time to start exploring AngularJS—components and models and views, oh my!—so that's next up on the docket. Until then, happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor. He has written more than 100 articles, is an F#MVP, has authored and coauthored a dozen books. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

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

Extreme Performance Linear Scalability

NCache™

Distributed Cache

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

NosDB

NoSQL Database

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

100% Native .NET

Open Source (Apache 2.0 License)

Microsoft Azure amazon web services™ Windows Server 2012 Visual Studio

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

MARK MICHAELIS

Essential MSBuild: A Build Engine Overview for .NET Tooling

Those of you who have been following .NET Core over the past few years (has it been that long?) know all too well that the "build system" has experienced a significant amount of flux, whether it be dropping built-in support for gulp or the demise of Project.json. For me as a columnist, these changes have been challenging as I didn't want you, my dear readers, to spend too much time learning about features and details that ultimately were only going to be around for a few months. This is why, for example, all my .NET Core-related articles were built on Visual Studio .NET 4.6-based *.CSPROJ files that referenced NuGet packages from .NET Core rather than actually compiled .NET Core projects.

This month, I'm pleased to report, the project file for .NET Core projects has stabilized into (would you believe) an MSBuild file. However, it's not the same MSBuild file from earlier Visual Studio generations, but rather an improved—simplified—MSBuild file. It's a file that (without getting into religious wars about curly vs. angle brackets) includes all the features of Project.json but with the accompanying tool support of the traditional MSBuild file we've come to know (and perhaps love?) since Visual Studio 2005. In summary, the features include open source, cross-platform compatibility, a simplified and human-editable format and, finally, full modern .NET tool support including wildcard file referencing.

Tooling Support

To be clear, features such as wild cards were always supported in MSBuild, but now the Visual Studio tooling works with them, as well. In other words, the most important news about MSBuild is that it's tightly integrated as the build system foundation for all the new .NET Tooling—DotNet.exe, Visual Studio 2017, Visual Studio Code and Visual Studio for Mac—and with support for both .NET Core 1.0 and .NET Core 1.1 runtimes.

The big advantage of the strong coupling between .NET Tooling and MSBuild is that any MSBuild file you create is compatible with all the .NET Tooling and can be built from any platform.

The .NET Tooling for MSBuild integration is coupled via the MSBuild API, not just a command-line process. For example, executing the .NET CLI command Dotnet.exe Build doesn't internally spawn the msbuild.exe process. However, it does call the MSBuild API in process to execute the work (both the MSBuild.dll and Microsoft.Build.* assemblies). Even so, the output—regardless of the tool—is similar across platforms because there's a shared logging framework with which all the .NET Tools register.

Figure 1 **A Basic Sample CSProj/MSBuild File**

```
<Project>
  <PropertyGroup>
    <TargetFramework>netcoreapp1.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <Compile Include="**\*.cs" />
    <EmbeddedResource Include="**\*.resx" />
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.NETCore.App">
      <Version>1.0.1</Version>
    </PackageReference>
    <PackageReference Include="Microsoft.NET.Sdk">
      <Version>1.0.0-*</Version>
      <PrivateAssets>All</PrivateAssets>
    </PackageReference>
  </ItemGroup>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
</Project>
```

*.CSPROJ/MSBuild File Structure

As I mentioned, the file format itself is simplified down to the bare minimum. It supports wildcards, project and NuGet package references, and multiple frameworks. Furthermore, the project-type GUIDs found in the Visual Studio-created project files of the past are gone.

Figure 1 shows a sample *.CSPROJ/MSBuild file.

Let's review the structure and capabilities in detail:

Simplified Header: To begin, notice that the root element is simply a Project element. Gone is the need for even the namespace and version attributes:

```
ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
```

(Though, they're still created in the release candidate version tooling.) Similarly, even the need for importing the common properties is merely optional:

```
<Import Project=
  "$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props" />
```

Project References: From the project file, you can add entries to the item group elements:

- NuGet Packages:

```
<PackageReference Include="Microsoft.Extensions.Configuration">
  <Version>1.1.0</Version>
</PackageReference>
```

- Project References:

```
<ProjectReference Include="..\ClassLibrary\ClassLibrary.csproj" />
```

- Assembly References:

```
<Reference Include="MSBuild">
  <HintPath>...</HintPath>
</Reference>
```

A direct assembly reference should be the exception as a NuGet reference is generally preferred.

Wildcard Includes: Compiled code files and resource files can all be included via wildcards.

```
<Compile Include="**\*.cs" />
<EmbeddedResource Include="**\*.resx" />
<Compile Remove="CodeTemplates\**" />
<EmbeddedResource Remove="CodeTemplates\**" />
```

However, you can still select specific files to ignore using the remove attribute. (Note that support for wildcards is frequently referred to as *globbing*.)

Multi-Targeting: To identify which platform you're targeting, along with the (optional) output type, you can use a property group with the TargetFramework elements:

```
<PropertyGroup>
  <TargetFramework>netcoreapp1.0</TargetFramework>
  <TargetFramework>netstandard1.3</TargetFramework>
</PropertyGroup>
```

Given these entries, the output for each target will be built into the bin\Debug or bin\Release directory (depending on which configuration you specify). If there's more than one target, the build execution will place the output into a folder corresponding to the target framework name.

No Project Type GUIDs: Note that it's no longer necessary to include a project-type GUID that identifies the type of project.

Visual Studio 2017 Integration

When it comes to Visual Studio 2017, Microsoft continues to provide a rich UI for editing the CSPROJ/MSBuild project file. **Figure 2**, for example, shows Visual Studio 2017 loaded with a CSPROJ file listing, slightly modified from **Figure 1**, that includes target framework elements for netcoreapp1.0 and net45, along with package references for Microsoft.Extensions.Configuration, Microsoft.NETCore.App, and Microsoft.NET.Sdk, plus an assembly reference to MSBuild, and a project reference to SampleLib.

Notice how each dependency type—assembly, NuGet package or project reference—has a corresponding group node within the Dependencies tree in Solution Explorer.

Furthermore, Visual Studio 2017 supports dynamic reloading of the project and solution. For example, if a new file is added to the project directory—one that matches one of the globbing wild cards—Visual Studio 2017 automatically detects the changes and displays the files within the Solution Explorer. Similarly, if you exclude a file from a Visual Studio project (via the Visual Studio menu option or the Visual Studio properties window), Visual Studio will automatically update the project file accordingly. (For example, it will add a <Compile Remove="CommandLine.cs" /> element to exclude the CommandLine.cs file from compiling within the project.)

Furthermore, edits to the project file will be automatically detected and reloaded into Visual Studio 2017. In fact, the Visual Studio project node within Solution Explorer now supports a built-in Edit <Project File> menu option that opens the project file within the Visual Studio edit window without first requesting that you unload the project.

There's built-in migration support within Visual Studio 2017 to convert projects to the new MSBuild format. If you accept its prompt, your project will automatically upgrade from a Project.json/*.XPROJ type to an MSBUILD/*.CSPROJ type. Note that such an upgrade will break backward compatibility with Visual Studio 2015 .NET Core projects, so you can't have some of your team working on the same .NET Core project in Visual Studio 2017 while others use Visual Studio 2015.

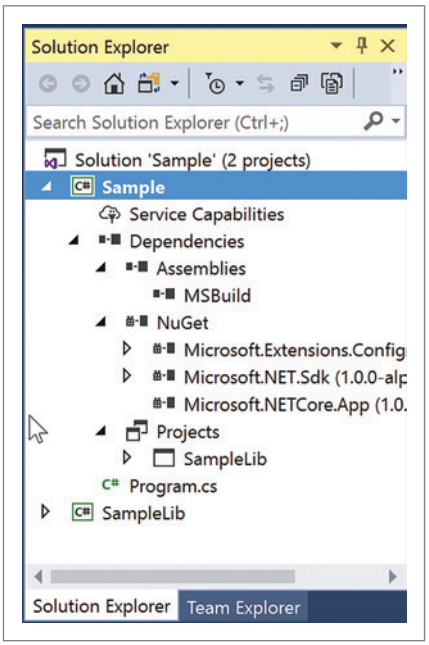


Figure 2 **Solution Explorer is a Rich UI on Top of a CSProj File**

MSBuild

I would be remiss to not point out that back in March 2016, Microsoft released MSBuild as open source on GitHub (github.com/Microsoft/msbuild) and contributed it to the .NET Foundation (dotnetfoundation.org). Establishing MSBuild as open source put it on track for platform portability to Mac and Linux—ultimately allowing it to become the underlying build engine for all the .NET Tooling.

Other than the CSPROJ\MSBuild file PackageReference element identified earlier, MSBuild version 15 doesn't introduce many additional features beyond open source and cross platform. In fact, comparing the command-line help shows the options are identical. For those not already familiar with it, here's a list of the most common options you should be familiar with from the general syntax *MSBuild.exe [options] [project file]*:

/target:<target>: Identifies the build target within the project file to execute along with any dependencies it might have (/t is the abbreviation).

/property:<n>=<v>: Sets or overrides any project properties (identified in the ProjectGroup element of a project file). For example, you can use the property to change the configuration or the output directory, as in /property:Configuration=Release;OutDir=bin\ (/p is the abbreviation).

/maxcpucount[:n]: Specifies the number of CPUs to use. By default, msbuild runs on a single CPU (single-threaded). If synchronization isn't a problem you can increase that by specifying the level of concurrency. If you specify the /maxcpucount option without providing a value, msbuild will use the number of processors on the computer.

/preprocess[:file]: Generates an aggregated project file by inlining all the included targets. This can be helpful for debugging when there's a problem.

@file: Provides one (or more) response files that contains options. Such files have each command-line option on a separate line (comments prefixed with "#"). By default, MSBuild will import a file named msbuild.rsp from the first project or solution built. The response file is useful for identifying different build properties and targets depending on which environment (Dev, Test, Production) you're building, for example.

Dotnet.exe

The dotnet.exe command line for .NET was introduced about a year ago as a cross-platform mechanism for generating, building and running .NET Core-based projects. As already mentioned, it has been updated so that it now relies heavily on MSBuild as the internal engine for the bulk of its work where this makes sense.

Here's an overview of the various commands:

dotnet new: Creates your initial project. Out of the box this project generator supports the Console, Web, Lib, MSTest and XUnitTest project types. However, in the future you can expect it to allow you to provide custom templates so you can generate your own project types. (As it happens, the new command doesn't rely on MSBuild for generating the project.)

dotnet restore: Reads through the project dependencies specified in the project file and downloads any missing NuGet packages and tools identified there. The project file itself can either be specified as an argument or be implied from the current directory (if there's more than one project file in the current directory, specifying which one to use is required). Note that because restore leverages the MSBuild engine for its work, the dotnet command allows for additional MSBuild command-line options.

dotnet build: Calls on the MSBuild engine to execute the build target (by default) within the project file. Like the restore command, you can pass MSBuild arguments to the dotnet build command. For example, a command such as dotnet build /property:configuration=Release will trigger a Release build to be output rather than a Debug build (the default). Similarly, you can specify the MSBuild target using /target (or /t). The dotnet build /t:compile command, for example, will run the compile target.

dotnet clean: Removes all the build output so that a full build rather than an incremental build will execute.

dotnet migrate: Upgrades a Project.json/*.XPROJ-based project into the *.CSPROJ/MSBuild format.

dotnet publish: Combines all the build output along with any dependencies into a single folder, thereby staging it for deployment to another machine. This is especially useful for self-contained deployment that includes not only the compile output and the dependent packages, but even the .NET Core runtime itself. A self-contained application doesn't have any prerequisites that a particular version of the .NET platform already be installed on the target machine.

dotnet run: Launches the .NET Runtime and hosts the project and/or the compiled assemblies to execute your program. Note that for ASP.NET, compilation isn't necessary as the project itself can be hosted.

There's a significant overlap between executing msbuild.exe and dotnet.exe, leaving you with the choice of which one to run. If you're building the default msbuild target you can simply execute the command "msbuild.exe" from within the project directory and it will compile and output the target for you. The equivalent dotnet.exe command is "dotnet.exe msbuild." On the other hand, if you're running a "clean" target the command is "msbuild.exe /t:clean" with MSBuild, versus "dotnet.exe clean" with dotnet. Furthermore, both tools support extensions. MSBuild has a comprehensive extensibility framework both within the project file itself and via .NET assemblies (see bit.ly/2flUBza). Similarly, dotnet can be extended but the recommendation for this, too, essentially involves extending MSBuild plus a little more ceremony.

While I like the idea of dotnet.exe, in the end it doesn't seem to offer much advantage over MSBuild except for doing the things that MSBuild doesn't support (of which dotnet new and dotnet run are perhaps the most significant). In the end, I feel that MSBuild allows you to do the simple things easily while still enabling the complicated stuff when needed. Furthermore, even the complicated stuff in MSBuild can be simplified down by providing reasonable defaults. Ultimately, whether dotnet or MSBuild is preferable comes down to preference, and time will tell which the development community generally settles on for the CLI front end.

global.json

While Project.json functionality has migrated to CSPROJ, global.json is still fully supported. The file allows specification of project directories and package directories, and identifies the version of the SDK to use. Here's a sample global.json file:

```
{
  "projects": [ "src", "test" ],
  "packages": "packages",
  "sdk": {
    "version": "1.0.0-preview3",
    "runtime": "clr",
    "architecture": "x64"
  }
}
```

The three sections correspond to the main purposes of the global.json file:

projects: Identifies the root directories in which .NET projects are located. The projects node is critical for debugging into the .NET Core source code. After cloning the source code, you can add the directory into the projects node and Visual Studio will then automatically load it up as a project within the solution.

packages: Indicates the location of your NuGet packages folder.

sdk: Specifies which version of the runtime to use.

Wrapping Up

In this article, I've provided a broad overview of all the places that MSBuild is leveraged within the .NET Tooling suite. Before closing, let me offer one bit of advice from my experience working on projects with thousands of lines of MSBuild. Don't fall into the trap of scripting too much in a declarative, loosely typed language like the XML MSBuild schema. That's not its purpose. Rather, the project files should be relatively thin wrappers that identify the order and dependencies among your build targets. If you let your MSBuild project file get too big, it can become a pain to maintain. Don't wait too long before you refactor it into something like C# MSBuild tasks that can be debugged and easily unit tested. ■

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

Thanks to the following IntelliTect technical experts for reviewing this article: *Kevin Bost, Grant Erickson, Chris Finlayson, Phil Spokas and Michael Stokesbary*

Spreadsheets Made Easy.

Fastest Calculations

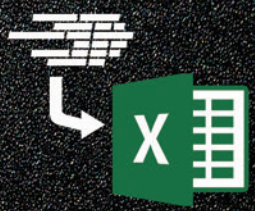
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Powerful Controls

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

Comprehensive Charting

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



Scalable Reporting

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

FRANK LA VIGNE

Exploring the UWP Community Toolkit

As the Universal Windows Platform (UWP) evolves, common UI patterns and challenges emerge. While the UWP provides a rich feature set of controls and APIs out of the box, there are some gaps in what's provided and some features require a lot of coding and effort. To address this gap and make UWP development faster and more accessible to beginners, Microsoft has created the UWP Community Toolkit and placed it on GitHub (bit.ly/2b1PAJY).

The goal of the UWP Community Toolkit is to engage the developer community to create a series of controls and helper functions to simplify UWP development. The UWP Community Toolkit has five categories of tools: Controls, Notifications, Animations, Services and Helpers. In this month's column, I'll walk through each of the categories and demonstrate how to implement the toolkit into your UWP projects.

UWP Community Toolkit Sample App

The best way to explore the UWP Community Toolkit is to download the UWP Community Toolkit Sample App from the Windows Store (bit.ly/2bypOyw). Not only does the app showcase all of the features of the toolkit (see **Figure 1**), but it also provides interactive demos of controls, animations, notifications and more, as shown in **Figure 2**.

The app will also provide XAML and code representations of the controls, thereby making it faster to tinker, tweak, and implement UI elements.

> The goal of the UWP Community Toolkit is to engage the developer community to create a series of controls and helper functions to simplify UWP development.

Interface Controls As previously stated, the UWP Community Toolkit contains a number of UI controls: 13 at press time. Many of these controls are commonly found in mobile applications or Web sites optimized for mobile. One such example is the HamburgerMenu control. By convention, the hamburger menu, which consists of three horizontal lines, indicates that there's a collapsed menu. Touching or clicking on the control toggles its state. **Figure 3** and **Figure 4** demonstrate what the control looks like collapsed and expanded, respectively.

Be sure to explore the features and settings of every control in the UWP Community Toolkit. They have an enormous degree of flexibility and are sure to add value to every one of your UWP projects.

Notifications Windows 10 provides a rich API that lets developers create live tiles and toast notifications to alert users of timely events and keep users up-to-date. For a full exploration of tiles and toast notifications, I recommend

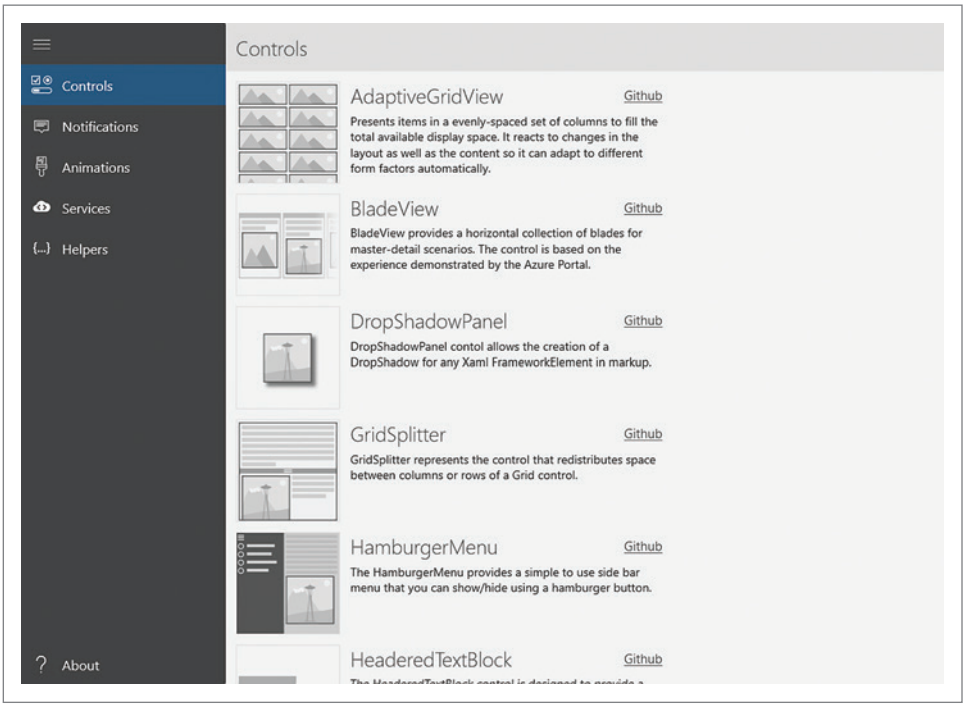


Figure 1 **Home Screen of the UWP Community Toolkit Sample App**

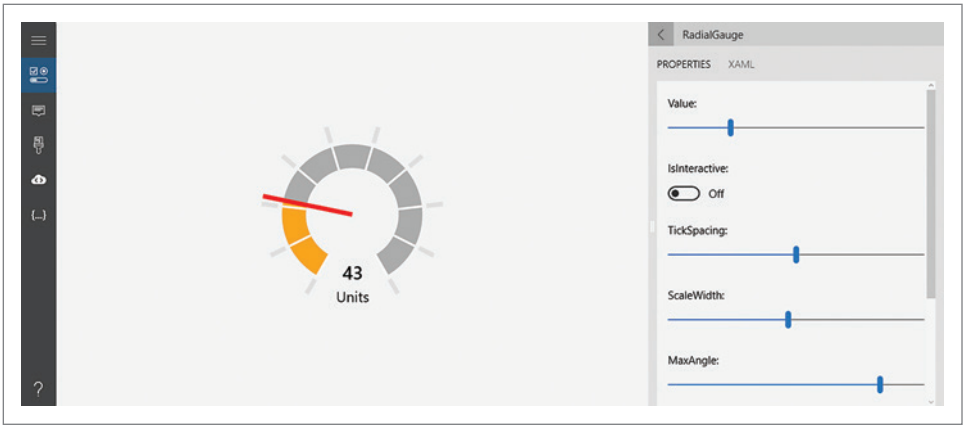


Figure 2 **Adjusting the Parameters of the RadialGauge Control**

reading the Tiles and Toasts blog that's written by the team that owns the UWP tile and toast APIs (bit.ly/2fzFsv1).

While useful and powerful, creating notifications and live tiles can prove challenging to developers unfamiliar with XML and XPath. The UWP Community Toolkit makes this easier by placing an object model on top of the underlying structure to generate the UI for toasts notifications and live tiles.

> Be sure to explore the features and settings of every control in the UWP Community Toolkit. They have an enormous degree of flexibility and are sure to add value to every one of your UWP projects.

Animations Windows 10 introduced the Composition API, a rich, declarative, retained-mode API that any UWP app can use to create composition objects, animations and visual effects directly. Essentially, it's a layer between XAML and the underlying DirectX architecture. The Composition API is meant to provide a way for XAML developers to enjoy many of the performance benefits of DirectX while writing code in C#. A full discussion of the Composition API is beyond the scope of this column, but you can find out more at bit.ly/2fzCfeQ.

The UWP Community Toolkit provides easy access to the power of this API by providing both XAML behaviors and extension methods on top of the Composition API. The UWP Community Toolkit Sample App even has a control surface to experiment with different parameters on a set of five animations. The toolkit will even generate the C# code and XAML markup to implement the animation.

Services We live in an era of the cloud and apps are better when they're connected to services in the cloud. However, adding these services to your apps can introduce a lot of complexity: implementing authentication via OAuth, working with REST APIs and the fluid nature of these APIs. Fortunately, the UWP Community Toolkit provides an extensive object model that deals with a lot of this complexity. Currently, the UWP Toolkit provides support for Bing, LinkedIn, Facebook, Twitter and the Microsoft Graph Service. This means that developers can easily connect to these online services with minimal effort.

Helper Functions The UWP Community Toolkit also includes various helper functions to make common app development tasks easier. For instance, before querying a cloud service, apps should

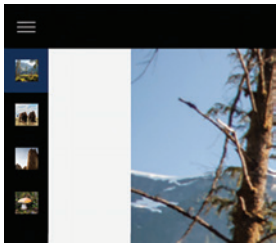


Figure 3 **A Collapsed Hamburger Menu**

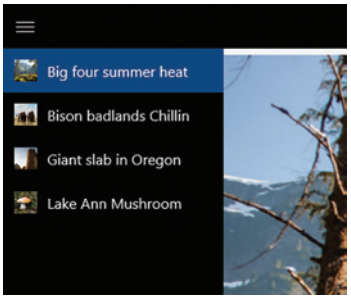


Figure 4 **An Expanded Hamburger Menu**

Figure 5 **UWP Community Toolkit NuGet Packages**

NuGet Package Name	Details
Microsoft.Toolkit.Uwp	Main NuGet package includes code-only helpers such as Colors conversion tool, Internet Connection detection, Storage file handling, a Stream helper class and so on.
Microsoft.Toolkit.Uwp.Notifications	Notifications Package—Generate tile, toast and badge notifications for Windows 10 via code. Includes IntelliSense support to avoid having to use the XML syntax.
Microsoft.Toolkit.Uwp.Notifications.Javascript	Notification Packages for JavaScript.
Microsoft.Toolkit.Uwp.Services	Services Package—this NuGet package includes the service helpers for Bing, Facebook, LinkedIn, Microsoft Graph and Twitter.
Microsoft.Toolkit.Uwp.UI	UI Packages—XAML converters, Visual tree extensions and helpers for your XAML UI.
Microsoft.Toolkit.Uwp.UI.Animations	Animations and Composition behaviors such as Blur, Fade, Rotate and so on.
Microsoft.Toolkit.Uwp.UI.Controls	XAML Controls such as RadialGauge, RangeSelector and so on.

Figure 6 **XAML to Create the Sample App's UI**

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid.RowDefinitions>
    <RowDefinition Height="33*"/>
    <RowDefinition Height="607*"/>
  </Grid.RowDefinitions>
  <Button Name="btnSearch" Click="btnSearch_Click" Grid.Row="0">Search Bing</Button>
  <ListView Name="lvSearchResults" Grid.Row="1">
    <ListView.ItemTemplate>
      <DataTemplate>
        <StackPanel>
          <TextBlock Text="{Binding Title}" FontWeight="Bold"></TextBlock>
          <TextBlock Text="{Binding Published}" FontStyle="Italic" ></TextBlock>
          <TextBlock Text="{Binding Link}" Foreground="Blue"></TextBlock>
          <TextBlock Text="{Binding Summary}"></TextBlock>
        </StackPanel>
      </DataTemplate>
    </ListView.ItemTemplate>
  </ListView>
</Grid>
```

check to see if there's a network connection. While that can be done without this toolkit, it can be done only in one line of code with the toolkit. Helpers are also available for printing, background tasks, obtaining system information and more.

Adding the UWP Community Toolkit to Your Project

The best way to add the UWP Community Toolkit to your project is through NuGet. Create a new blank UWP project in Visual Studio by choosing New Project from the File menu. Select Installed | Templates | Windows | Universal | Blank App (Universal Windows). Name the project UWPToolKitApp and then click OK.

The easiest way to add the UWP Community Toolkit to your project is to add the NuGet packages. In Solution Explorer, right-click on References and choose Manage NuGet Packages from the context menu to show the NuGet Package Manager.

In the search box, type Microsoft.Toolkit.Uwp to bring up all the NuGet packages associated with the UWP Community Toolkit. You should see a handful of results that include the UWP Community Toolkit itself, plus other packages and libraries. For a quick reference chart on which packages contain which functionality, refer to **Figure 5**.

> While useful and powerful, creating notifications and live tiles can prove challenging to developers unfamiliar with XML and XPath.

For now, you'll want to add the main NuGet Package (Microsoft.Toolkit.Uwp) and the Services package (Microsoft.Toolkit.Uwp.Services). Click on the entry for Microsoft.Toolkit.Uwp and then on the Install button that appears toward the right side of the NuGet Package Manager. If prompted with a Review Changes dialog, review the changes and then click OK. Clicking Cancel here will cancel the NuGet Package install. You might also be shown a License Acceptance dialog; click I Accept to accept the license terms. You have an option to review the license. You also have the option to decline the license. Clicking I Accept will complete the package install. Repeat the process for the Microsoft.Toolkit.Uwp.Services package, which might have multiple license terms to accept.

Once these tasks are completed, you should see the NuGet packages in the References section of your project in Solution Explorer.

Searching with Bing Now, you're going to create a sample app that will check for online connectivity and the device model on which the app is running. It will then present you the option to search Bing for information about that device.

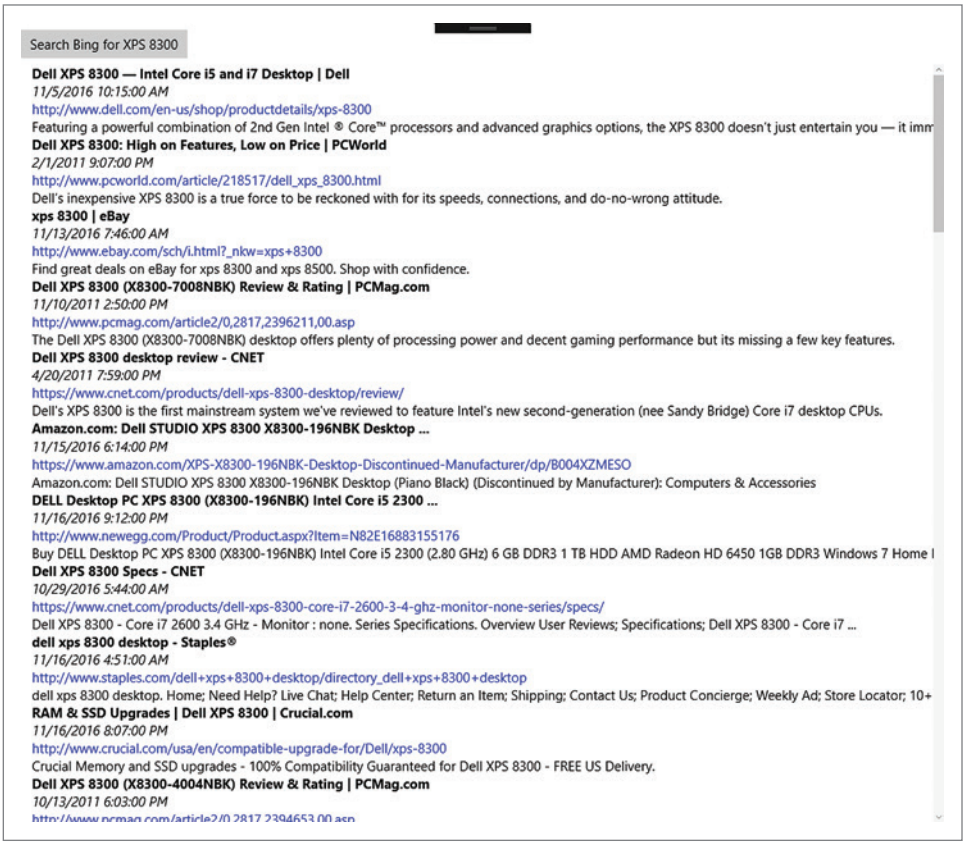


Figure 7 **Search Results for Device Model**

Figure 8 **Code to Create Sample Toast Notification Content**

```
private ToastContent CreateDummyToast()
{
  return new ToastContent()
  {
    Launch = "action=viewEvent&eventId=1983",
    Scenario = ToastScenario.Default,

    Visual = new ToastVisual()
    {
      BindingGeneric = new ToastBindingGeneric()
        {
          Children =
            {
              new AdaptiveText()
                {Text = $"Bing search results returned for
                  {SystemInformation.DeviceModel}"},
              new AdaptiveText()
                {Text = $"Available Memory {SystemInformation.AvailableMemory}"},
              new AdaptiveText(){Text =
                $"Running {SystemInformation.OperatingSystem}
                {SystemInformation.OperatingSystemVersion}
                ({SystemInformation.OperatingSystemArchitecture})"}
            }
        }
      }
    };
  }
```

Open the MainPage.xaml file and add the XAML in **Figure 6** to create a simple UI.

In the MainPage.xaml.cs file, add the following code to the constructor method:

```
btnSearch.IsEnabled = ConnectionHelper.IsInternetAvailable;
btnSearch.Content = $"Search Bing for {SystemInformation.DeviceModel}";
```

And add the following code to the MainPage.xaml.cs file:

```
private async void btnSearch_Click(object sender, RoutedEventArgs e)
  {
    var searchConfig = new BingSearchConfig
    {
      Country = BingCountry.UnitedStates,
      Language = BingLanguage.English,
      Query = SystemInformation.DeviceModel,
      QueryType = BingQueryType.Search
    };

  lvSearchResults.ItemsSource =
    await BingService.Instance.RequestAsync(searchConfig, 50);
  }
```

Visual Studio will help you add the appropriate using statements to your code.

Run the solution now. If your device has Internet connectivity, then the Search button should be enabled. You'll also see the device model in the button text. If the button is enabled, click on it and you'll see something similar to **Figure 7**. Naturally, the specific results vary based on the device the app runs on.

Toast Notifications Another feature of the UWP Community Toolkit that simplifies common developer tasks is toast notifications. Toast notifications are an important part of the UX in UWP apps, because they keep users informed as to the state of the app. However, the Toast Notification API requires thorough knowledge of XML. This can confuse novice-level developers and even frustrate more seasoned ones. The UWP Community Toolkit really simplifies creating toast notifications.

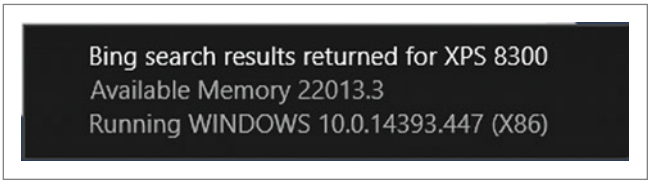


Figure 9 **Toast Notification Returned**

For the sample app, you'll add a simple toast notification to inform the user that the search results have returned. This might not add much value to the sample app, as Bing returns results instantly. However, this would be useful in alerting the user to the completion of a long-running task.

The first step is to add the Microsoft.Toolkit.Uwp.Notifications NuGet package to the project. Repeat the steps mentioned earlier to add it to the project.

Next, add the code in **Figure 8** to the MainPage.xaml.cs file to create the content of the Toast notification. It uses the System Information helper class to fill in some details about the device on which the app is running.

Looking at the code more closely, you'll see that the UWP Community Toolkit adds an object model around the Toast Notification API XML format. This simplifies the process of creating content for the toast notification. Developers can enjoy the power and flexibility of the open-ended format while getting an object model (and IntelliSense) to write code against.

Now, add the following method:

```
private void PopToast()
{
  ToastContent toastContent = CreateDummyToast();
  ToastNotificationManager.CreateToastNotifier()
    .Show(new ToastNotification(toastContent.GetXml()));
}
```

The first line calls the CreateDummyToast method that builds the content of the toast notification and assigns it to a ToastContent object. The second line uses the GetXml method to convert the object model to the XML format that the Toast Notification API expects.

All that's left to do now is call the PopToast method right after the search results return from Bing in the btnSearch_Click event handler. Run the solution and, just as before, click on the Search button. Almost immediately, you'll see a notification similar to **Figure 9** appear.

Wrapping Up

The UWP Community Toolkit provides essential resources for creating rich and engaging UWP apps. From new controls to cloud services API libraries, it enables developers to easily create apps that are connected to services in the cloud without having to deal with lower-level plumbing concerns of REST APIs. Best of all, thanks to the community of developers contributing to it, the UWP Community Toolkit is getting better all the time. The feature set is continually growing and the team is accepting contributions. It's truly something created for the community by the community. ■

Frank La Vigne *is an independent consultant, where he helps customers leverage technology in order to create a better community. He blogs regularly at FranksWorld.com and has a YouTube channel called Frank's World TV (FranksWorld.TV).*

Thanks to the following technical experts for reviewing this article: *David Catuhe*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LAS VEGAS
MAR 13-17 2017
BALLY'S, LAS VEGAS, NV

EVENT PARTNERS

Microsoft Magenic

SUPPORTED BY

Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

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

Track Topics include:

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

TURN THE PAGE FOR FULL AGENDA DETAILS

Sunday Pre-Con Hands-On Labs

Choose From:

- Angular
- Azure
- XAML

NEW! ONLY $595

SPACE IS LIMITED

Register by January 20 and Save $400!*

Use promo code VSLJAN4

REGISTER NOW

ROCK YOUR CODE TOUR 2017

LAS VEGAS	AUSTIN	WASH. DC	REDMOND	CHICAGO	ANAHEIM	ORLANDO
MARCH 13-17	MAY 15-18	JUNE 12-15	AUG 14-18	SEPT 18-21	OCT 16-19	NOV 13-17

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

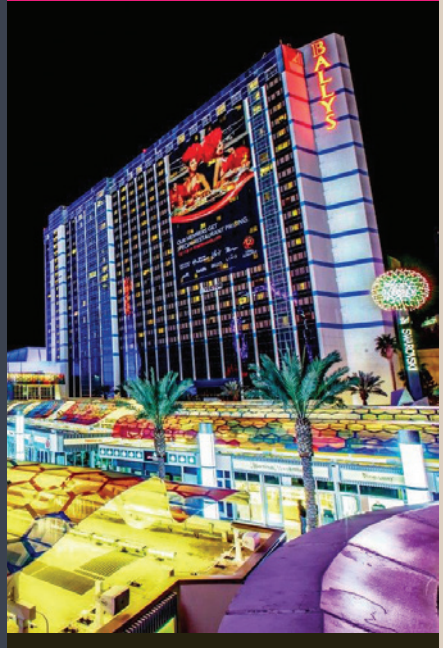
linkedin.com – Join the "Visual Studio Live" group!

vslive.com/lasvegasmsdn

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

AGENDA AT-A-GLANCE

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



CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

REGISTER NOW

ALM / DevOps | Cloud Computing | Database and Analytics | Mobile Client | Software Practices

START TIME	END TIME	
		Full Day Hands-On Labs: Sunday, March 12, 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	**HOL01** Full Day Hands-On Lab: Build an Azure App in a Day - *Brian Randell*
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas
2:00 PM	6:00 PM	Workshop Continues
		Pre-Conference Workshops: Monday, March 13, 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: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries & Roy Cornelissen*
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas
2:00 PM	6:00 PM	Workshop Continues
7:00 PM	9:00 PM	Dine-A-Round
		Day 1: Tuesday, March 14, 2017
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:00 AM	Keynote: To Be Announced Donovan Brown, Senior DevOps Program Manager, US Developer Division Team,
9:15 AM	10:30 AM	**T01** Essential Web Development with ASP.NET Core - *Mark Michaelis* / **T02** An Overview of the Xamarin Programming Platforms - *Laurent Bugnion*
10:45 AM	12:00 PM	**T06** Migrating to ASP.NET Core - A True Story - *Adam Tuliper* / **T07** Building Truly Universal Applications with Windows, Xamarin and MVVM - *Laurent Bugnion*
12:00 PM	1:00 PM	Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors
1:30 PM	2:45 PM	**T11** An Introduction to TypeScript - *Jason Bock* / **T12** What's New for Developers in SQL Server 2016 - *Leonard Lobel*
3:00 PM	4:15 PM	**T16** Assembling the Web - A Tour of WebAssembly - *Jason Bock* / **T17** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*
4:15 PM	5:30 PM	Welcome Reception
		Day 2: Wednesday, March 15, 2017
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**W01** Angular 101: Part 1 - *Deborah Kurata* / **W02** Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - *Laurent Bugnion*
9:30 AM	10:45 AM	**W06** Angular 101: Part 2 - *Deborah Kurata* / **W07** Busy .NET Developer's Guide to Native iOS - *Ted Neward*
11:00 AM	12:00 AM	General Session: To Be Announced
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)
1:30 PM	2:45 PM	**W11** User Authentication for ASP.NET Core MVC applications - *Brock Allen* / **W12** Cloud Enable an Existing WPF LOB App - *Robert Green*
3:00 PM	4:15 PM	**W16** Securing Web APIs in ASP.NET Core - *Brock Allen* / **W17** Strike Up a Conversation with Cortana on Windows 10 - *Walt Ritscher*
4:30 PM	5:45 PM	**W21** ASP.NET Core 1.0 Tag Helpers - *Robert Boedigheimer* / **W22** Busy Developer's Guide to NoSQL - *Ted Neward*
7:00 PM	8:30 PM	Experience The LINQ Vortex & High Roller Event
		Day 3: Thursday, March 16, 2017
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**TH01** Debugging Your Website with Fiddler and Chrome Developer Tools - *Robert Boedigheimer* / **TH02** Windows for Makers: Raspberry Pi, Arduino & IoT - *Nick Landry*
9:30 AM	10:45 AM	**TH06** I Say A "Front-end Build Pipeline", You Say WAT!? - *Chris Klug* / **TH07** Building Cross-Platform Business Apps with CSLA .NET - *Rockford Lhotka*
11:00 AM	12:15 PM	**TH11** JavaScript Patterns for the C# Developer - *Ben Hoelting* / **TH12** Building Connected and Disconnected Mobile Apps - *James Montemagno*
12:15 PM	1:45 PM	Lunch
1:45 PM	3:00 PM	**TH16** Integrating AngularJS & ASP.NET MVC - *Miguel Castro* / **TH17** Native iOS and Android Development with C# and Xamarin - *James Montemagno*
3:15 PM	4:30 PM	**TH21** Increase Website Performance and Search with Lucene.Net Indexing - *Ben Hoelting* / **TH22** Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - *Nick Landry*
		Post-Conference Workshops: Friday, March 17, 2017 *(Separate entry fee required)*
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries
8:00 AM	5:00 PM	**F01** Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*

Speakers and sessions subject to change

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with Magenic

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

Visual Studio / .NET Framework	Web Client	Web Server	Windows Client	Modern Apps Live!

Full Day Hands-On Labs: Sunday, March 12, 2017 *(Separate entry fee required)*

Pre-Conference Workshop Registration - Coffee and Morning Pastries

HOL02 Full Day Hands-On Lab: AngularJS 2 - *Ted Neward*

HOL03 Full Day Hands-On Lab: An Introduction to Building XAML Applications - *Billy Hollis*

Lunch @ Le Village Buffet, Paris Las Vegas

Workshop Continues | Workshop Continues

Pre-Conference Workshops: Monday, March 13, 2017 *(Separate entry fee required)*

Pre-Conference Workshop Registration - Coffee and Morning Pastries

M02 Workshop: Developer Dive into SQL Server 2016 - *Leonard Lobel*

M03 Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka & Jason Bock*

M04 Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*

Lunch @ Le Village Buffet, Paris Las Vegas

Workshop Continues | Workshop Continues | Workshop Continues

Dine-A-Round

Day 1: Tuesday, March 14, 2017

Registration - Coffee and Morning Pastries

Microsoft

T03 Developer Productivity in Visual Studio 2017 - *Robert Green*

T04 Understanding the VR/AR Landscape - *Katherine Harris*

T05 Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*

T08 Roll Your Own Dashboard in XAML - *Billy Hollis*

T08 DevOps and Azure with the MS Stack - *Abel Wang*

T10 Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*

Lunch

Dessert Break - Visit Exhibitors

T13 A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*

T14 To Be Announced

T15 Architecture: The Key to Modern App Success - *Brent Edwards*

T18 Essential C# 7.0 - *Mark Michaelis*

T19 To Be Announced

T20 Focus on the User Experience #FTW - *Anthony Handley*

Welcome Reception

Day 2: Wednesday, March 15, 2017

Registration - Coffee and Morning Pastries

W03 What's New in Azure IaaS v2 - *Eric D. Boyd*

W04 Application Lifecycle Management (ALM) - *Brian Randell*

W05 DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*

W08 Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*

W09 Use Visual Studio to Scale Agile in Your Enterprise - *Richard Hundhausen*

W10 Mobile Panel - *James Montemagno, Ryan J. Salva, Kevin Ford, Rockford Lhotka*

General Session: To Be Announced

Birds-of-a-Feather Lunch

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

W13 I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - *Adam Tuliper*

W14 Professional Scrum Development Using Visual Studio 2017 - *Richard Hundhausen*

W15 C# Everywhere: How CSLA .NET Enables Amazing Cross-Platform Code Reuse - *Rockford Lhotka*

W18 Cloud Oriented Programming - *Vishwas Lele*

W19 Introduction to Containers and Docker - *Marcel de Vries*

W20 Coding for Quality and Maintainability - *Jason Bock*

W23 Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*

W24 Using Docker on Windows in VSTS Build and Release Management - *Marcel de Vries*

W25 Modern Mobile Development: Build a Single App For iOS & Android with Xamarin Forms - *Kevin Ford*

Experience The LINQ Vortex & High Roller Event

Day 3: Thursday, March 16, 2017

Registration - Coffee and Morning Pastries

TH03 Accelerate Your Mobile App Development with Azure App Services Mobile Apps - *Brian Noyes*

TH04 Agile: You Keep Using That Word - *Philip Japikse*

TH05 Modern Web Development: ASP.NET MVC and Web API - *Allen Conway*

H08 Connect All The Things with Azure Service Bus, Notification Hubs, Event Hubs, and IoT Hubs - *Brain Noyes*

TH09 Visualizing the Backlog with User Story Mapping - *Philip Japikse*

TH10 Modern Web Development: Building a Smart Web Client with TypeScript and Angular2 - *Allen Conway*

TH13 Add A Conversational User Interface to Your App with the Microsoft Bot Framework - *Walt Ritscher*

TH14 End-to-End Dependency Injection & Testable Code - *Miguel Castro*

TH15 Cloud Panel - *Rockford Lhotka*

Lunch

TH18 Introduction to R and Microsoft R Server - *James McCaffrey*

TH19 Open Source Software for Microsoft Developers - *Rockford Lhotka*

TH20 Universal Windows Development: UWP for PC, Tablet & Phone - *Nick Landry*

TH23 Introduction to Azure Machine Learning - *James McCaffrey*

TH24 SOLID - The Five Commandments of Good Software - *Chris Klug*

TH25 Using All That Data: Power BI to the Rescue - *Scott Diehl*

Post-Conference Workshops: Friday, March 17, 2017 *(Separate entry fee required)*

Post-Conference Workshop Registration - Coffee and Morning Pastries

F02 Workshop: Application Lifecycle Management (ALM) - *Brian Randell*

F03 Workshop: Modern App Deep Dive: Xamarin, Responsive Web, UWP, CSLA .NET - *Kevin Ford, Jason Bock, Brent Edwards, Allen Conwa*

vslive.com/lasvegasmsdn

For the Defense

I find that one of the coolest things about teaching is meeting interesting people and introducing them to you. To Apollo program computer designer Hugh Blair-Smith (msdn.com/magazine/mt703442) and rescue leader Gisli Olafsson (msdn.com/magazine/dn818503), I now add defense attorney Owen Walker.

Owen was a student in one of my earliest classes, back in 1994, (16-bit Windows SDK in C, with Charles Petzold as a text). Harvard Extension School is proud of its broad reach, so it didn't surprise me that he was a lawyer rather than a programmer. (I'll save the French diplomat story for another day.) He joined the class because he wanted to write an add-in for Microsoft Word to help him fill out his expense reports. He didn't know C, but had just enough Pascal to get confused (:= versus = for assignment, = versus == for comparison). He worked so hard I don't even want to think about it, even now 20-plus years later, and earned his A.

> Over beer one night after class, I asked why he spent his professional life defending bad guys, most of whom are guilty as heck.

Owen worked in the Federal Public Defender Office in Boston. Over beer one night after class, I asked why he spent his professional life defending bad guys, most of whom are guilty as heck. He said, "Yes, but some really are innocent, others, perhaps guilty of some moral infraction, are not technically guilty of a crime. But it's bigger than that: I don't want the cops just kicking down any door that they want to and dragging anyone away. I don't mind seeing bad guys caught and locked up, but I need to see it done by the book. If the prosecution can legally find evidence to prove guilt to a jury beyond reasonable doubt, fine. My job is to make sure they all do their jobs. Not for the benefit of the defendants, but for all of society."

Owen later became the Federal Public Defender for Boston, the constitutional official, paid for by your taxes and mine, responsible for getting defendants a fair shake. That position is often unpopular, sometimes quite deeply. In the performance of those constitutional duties, Owen defended shoe bomber Richard Reid (see nyti.ms/2fuTv7E). He retired in 2005.

I called him up recently to hear his thoughts on the tussles playing out today between collective security and individual privacy. For example, after the San Bernardino shootings last spring, the FBI captured the main shooter's iPhone and wanted to examine its contents. Unfortunately, the phone's internal storage was encrypted and the FBI couldn't easily crack it. So they went to court, trying to force Apple to do so, which Apple strongly resisted (see bit.ly/2eXAvux).

Owen disapproves of Apple's resistance, writing in a recent article that, "[Apple's] refusal to help the government decrypt the San Bernardino cell phone was, in my view, shocking, in view of the fact that getting access to the information might have revealed plans for ongoing violent terrorism." That particular case became moot when the FBI finally did manage to crack it with help from some serious geeks (see bit.ly/2f1IGJJ), but the underlying dilemma remains unresolved.

Apple and Google are now said to be developing communication security that even they can't break, even if they wanted to. That concerns many people, who think that the companies should be required to incorporate a back door open for lawful government access—requiring a warrant, issued by a judge, under strict guidelines.

Owen is one of these concerned parties. He writes that "lawful wiretapping authorized by a judge has been a key element to public safety, and has resulted in a vast number of serious criminals going to prison. Apple and Google, by absolute encryption, are making wiretapping impossible."

It's easy to respond with the knee-jerk, "When encryption is outlawed, only outlaws will encrypt," or, "Fear the government that fears your phone." And the slippery slope problem is obvious. But when a guy who puts it on the line to defend bad guys, not for their sake but for ours and our country's, thinks otherwise, we should probably at least give his ideas a good look. ■

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.*

Enterprise-Proven Distributed Caching

Trusted for over a decade, the easiest most powerful in-memory data grid to scale your .NET applications

- **Fast and linearly scalable**
- **Enterprise-grade availability**
- **Industry-leading ease of use**
- **Integrated in-memory computing**

Replacing AppFabric Caching?
Having Trouble Scaling Redis?

Try our AppFabric-compatible drop-in.
www.scaleoutsoftware.com/appfabric
Special migration pricing available.

Brought to you by the scalability architects

Step up to a battle-tested in-memory data grid that has been hardened by over 425 enterprise customer deployments. ScaleOut's technology makes advanced features such as parallel LINQ query and integrated MapReduce accessible to any .NET developer. Automatic configuration and turnkey global data replication deliver legendary ease-of-use. ScaleOut's world-class support meets the needs of mission-critical applications. Run on premises or in the cloud on Microsoft Azure or Amazon AWS.

ScaleOut Software

Download your free trial today!
www.scaleoutsoftware.com/trial

Please visit scaleoutsoftware.com to learn how ScaleOut's technology can help accelerate your business or call us at +1.503.643.3422

SYNCFUSION'S UNLIMITED FLAT-FEE LICENSE

WHY PAY FOR EXPENSIVE SOFTWARE DEVELOPMENT COSTS WHEN YOU CAN HAVE IT ALL WITH ONE LOW, ANNUAL FEE!

- No user count
- 800+ pre-built controls and frameworks
- Big Data, Dashboard, Data Integration, and Report Platforms
- Truly unlimited use
- Starting at **$3,995**

START YOUR EVALUATION TODAY!

www.syncfusion.com/MSDNunlimited

Syncfusion®