

msdn

magazine



Cross-Platform Development
with .NET Core.....20

When Only the Best Will Do

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



Free 30-day Trial
devexpress.com/try

Unleash the **UI Superhero** in You

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



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

msdn magazine



Cross-Platform Development
with .NET Core.....20

.NET Goes Cross-Platform with .NET Core
Phillip Carter and Zlatko Knezevic20

Microsoft Pushes C++ into the Future
Kenny Kerr24

Using Ionic and TACO to Create
Cross-Platform Mobile Apps
Adam Tuliper32

Data Processing and Machine
Learning on Spark
Eugene Chuvyrov40

Develop an Azure-Connected IoT Solution
in Visual Studio with C++
Marc Goodner.....52

COLUMNS

UPSTART

Ordinary People
Krishnan Rangachari, page 6

CUTTING EDGE

Pushing Notifications
to Mobile Apps
Dino Esposito, page 8

DATA POINTS

Handling the State of
Disconnected Entities in EF
Julie Lerman, page 14

ESSENTIAL .NET

Logging with .NET Core
Mark Michaelis, page 64

MODERN APPS

Writing UWP Apps for the
Internet of Things
Frank La Vigne, page 70

DON'T GET ME STARTED

Gods and Fools
David Platt, page 80



Break free from Oracle

**Now is the time to move your Oracle
databases to SQL Server with free licenses.***

Get the one data solution that has everything you need—
mission-critical performance, business intelligence, and
advanced analytics—all built in.

As the new industry leader in data, SQL Server has no equal.

Limited Time Offer. Learn more.
www.microsoft.com/break-free
1-800-867-3163

*Software Assurance subscription required



Visual Studio Marketplace

marketplace.visualstudio.com

Powered by



BUILD | JOIN | PUBLISH | SELL

Publish your extensions to the Marketplace. Join the Visual Studio Partner Program!

JOIN NOW!



Benefits & Resources to support your business.



Our BASIC level is always free!



Limited-time offer*: Paid levels get 30% off 1st year!

Join at
vspartner.com



JAVA TOOLS CHALLENGE



Help make Java great with Visual Studio Team Services and win your share of **\$90K in prizes!**

Are you up for the Challenge?

1. Create a Visual Studio Team Services (VSTS) extension that helps developers create, test, and/or deploy Java apps;
2. OR Create a Java app using either the VSTS Eclipse plugin (aka Team Explorer Everywhere) or our JetBrains IntelliJ plugin.

Register at
aka.ms/javatc



BUILD 2016

SAN FRANCISCO, CA
March 30 - April 1



Check out these Visual Studio partners at the Build 2016 Expo!



Find VS Partners at
Build 2016
aka.ms/buildvs



* Join the Visual Studio Partner Program before June 30, 2016 and receive \$2,000/\$3,000 off the first year of your Alliance/Premier-level membership.

General Manager Jeff Sandquist

Director Dan Fernandez

Editorial Director Mohammad Al-Sabt mmeditor@microsoft.com

Site Manager Kent Sharkey

Editorial Director, Enterprise Computing Group Scott Bekker

Editor in Chief Michael Desmond

Features Editor Sharon Terdeman

Features Editor Ed Zintel

Group Managing Editor Wendy Hernandez

Senior Contributing Editor Dr. James McCaffrey

Contributing Editors Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt, Bruno Terkaly

Vice President, Art and Brand Design Scott Shultz

Art Director Joshua Gould



President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

ART STAFF

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

PRODUCTION STAFF

Print Production Coordinator Lee Alexander

ADVERTISING AND SALES

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

ONLINE/DIGITAL MEDIA

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

LEAD SERVICES

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

MARKETING

Chief Marketing Officer Carmel McDonagh
Vice President, Marketing Emily Jacobs
Senior Manager, Marketing Christopher Morales
Marketing Coordinator Alicia Chew
Marketing & Editorial Assistant Dana Friedman

ENTERPRISE COMPUTING GROUP EVENTS

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



Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Vice President & Chief Financial Officer
Michael Rafter

Executive Vice President
Michael J. Valenti

Chief Technology Officer
Erik A. Lindgren

Chairman of the Board
Jeffrey S. Klein

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

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

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

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

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

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

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

Reaching the Staff

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

E-mail: To e-mail any member of the staff, please use the following form: FirstInitialLastName@1105media.com
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618

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

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





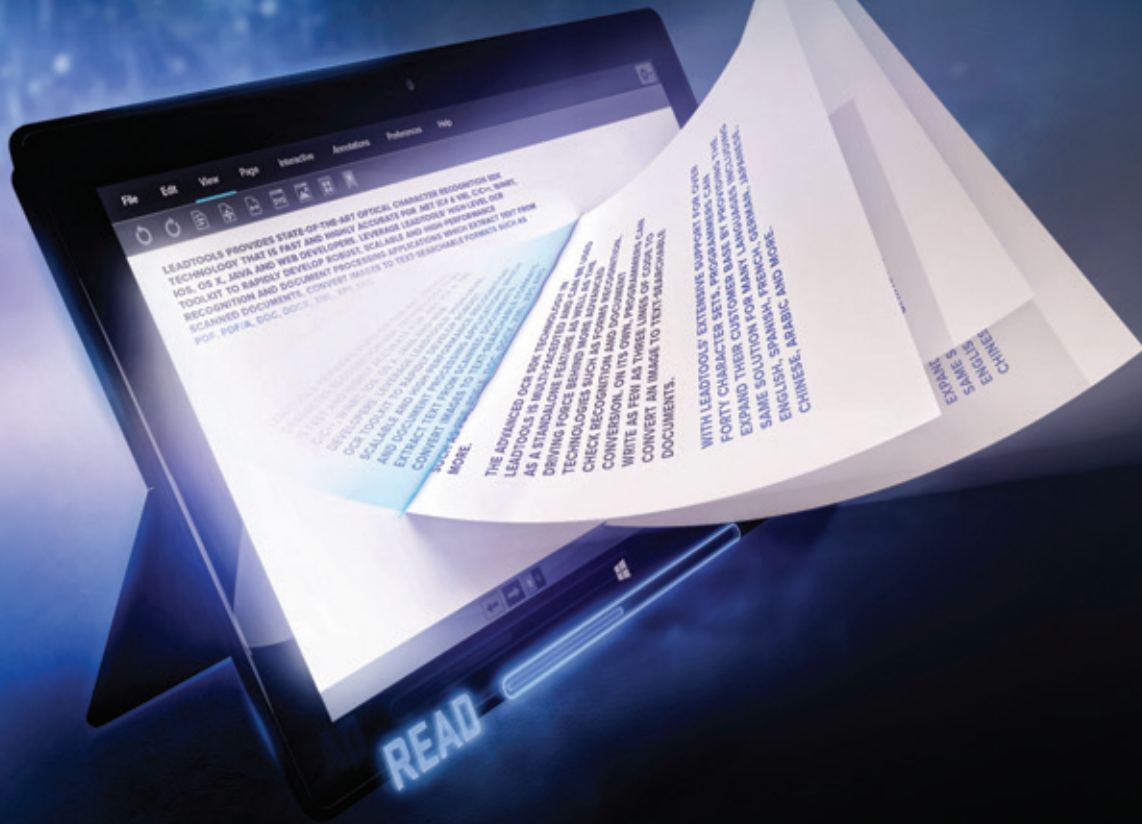
DESKTOP



TABLET



MOBILE



OCR FOR ANY CLIENT, DEVICE OR SERVER. DONE.

LEADTOOLS OCR SDK

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

Utilize multiple cores for unparalleled performance

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

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

.NET Windows API WinRT Linux iOS OS X Android HTML5





Build's Bold Direction

On March 30 Microsoft holds its annual Build conference in San Francisco, providing updates and guidance on a host of development technologies, platforms and strategies. In a sense, it's the first day of the new year on the Microsoft developer calendar.

No surprise, Xamarin and its pending acquisition by Microsoft are top topics at Build. Developers can expect to learn more about Microsoft's plans to improve cross-platform mobile development by leveraging and integrating Xamarin's expansive suite of technologies. As Forrester Research Inc. Vice President and Principal Analyst Jeffrey Hammond noted after the announcement, Microsoft has developed a robust portfolio of mobile infrastructure services that tie into the Microsoft Azure public cloud. With the addition of Xamarin, Microsoft now has a front-end mobile app dev story to match (see bit.ly/1oW6CQC).

Developers can expect to learn more about Microsoft's plans to improve cross-platform mobile development by leveraging and integrating Xamarin's expansive suite of technologies.

In fact, the Xamarin buy is just the latest in a parade of cross-platform-minded efforts from Microsoft. Prominent among them: .NET Core, which is comprised of a modular and cross-platform runtime and set of APIs that allow you to build cross-platform Web apps, libraries and console apps. You can expect Microsoft to have

plenty to say about .NET Core at Build. As authors Phillip Carter and Zlatko Knezevic describe in their feature article this month, .NET Core opens new opportunities for developers, enabling scenarios where apps and services can be deployed and run across Windows, Mac and Linux machines.

"Developers are genuinely excited about being able to develop and deploy on their OS of choice," says Carter, a program manager on the .NET Team at Microsoft. "Many places have a need to support more than just Windows as a deploy target. Furthermore, we believe .NET Core is an attractive offering to developers who aren't traditionally focused on Windows and .NET."

Microsoft has also been busy supporting cross-platform hybrid app development, by way of Visual Studio Tools for Apache Cordova (aka TACO). In his article, "Using Ionic and TACO to Create Cross-Platform Mobile Apps," Adam Tuliper describes how developers can create compelling UIs for their hybrid apps. He also looks at the new Cordova Tools extensions for IntelliSense and debugging in Visual Studio Code.

Finally, Eugene Chuvyrov, a cloud solutions architect at Microsoft, offers a glimpse at how Microsoft is reaching across platforms in the Big Data space. His feature, "Data Processing and Machine Learning on Spark," explores how the Linux-based, open source Spark framework powers cutting-edge analytics and machine learning in Azure and Visual Studio.

This is kind of amazing when you think about it. As Chuvyrov noted to me, "I think the overlooked fact is that an open source project (Hadoop, Spark) is running as a managed service in Azure with SLAs and enterprise support."

Microsoft has come a long way over the last 10 years. It'll be interesting to see how much further the company goes before Build 2017 rolls around.

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

© 2016 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN and Microsoft logos are used by 1105 Media, Inc. under license from owner.



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

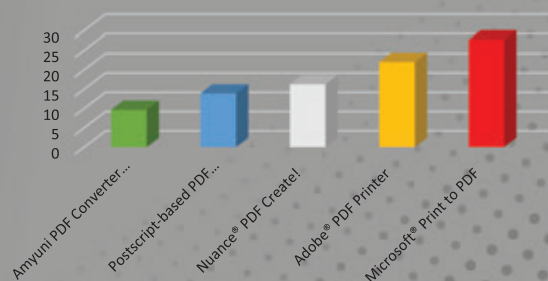
- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML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



AMYUNI 
Technologies

USA and Canada

Toll Free: 1866 926 9864

Support: 514 868 9227

sales@amyuni.com

Europe

UK: 0800-015-4682

Germany: 0800-183-0923

France: 0800-911-248

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

All development tools available at

www.amyuni.com

Ordinary People

Throughout my career, I've had a desire to be special. I've wanted to be the best, the youngest, the smartest or the hardest-working in everything I've ever done.

But I came to notice something odd. I would briefly enjoy the thrill of my achievements, and then brush them off casually—as if I didn't really do them. It wasn't humility. At some level, I felt like a ghost living someone else's life. I couldn't actually identify with this guy inside me who received all the accolades.

I grew to understand that I sought to be extraordinary because I felt broken. The only way I couldn't feel worthless was to be significant.

I realized also that if I could somehow *not* feel broken, then I wouldn't have to work so hard trying to be special. I came to wonder if I could embrace my own ordinariness; to that end, I've been trying a few experiments.

In stepping off the career gas pedal, I see that I'm so divorced from reality that I'm medicating all of my life's pain—family issues, relationship problems, existential angst—with work.

First: I take on projects that aren't as exciting or high-profile at work. They're projects I'm not immediately drawn to, but where my skills and expertise are needed. I focus simply on helping others in such projects.

When I do this, I tell my mind I can take joy in any job, that I'm not seeking glory. This begins to confuse my mind! It starts to wonder, "If I'm not seeking glory, does that mean I'm not broken?"

Second: I experiment with humility and silence in new ways. I let someone else present a project if they want to (and I love to present). When I feel the urge to destroy someone's argument, I try hard to say nothing at all.

Third: I step off the hamster wheel of over-achievement. Instead of trying to live up to a mythical ideal of a high-achiever, I take care of my body and actively try to work less. It's easy for me to get carried away with work. I can *control* work—if I get X amount

of work done, I can see Y amount of results—so I seek refuge in it, because I can't control life.

In stepping off the career gas pedal, I see that I'm so divorced from reality that I'm medicating all of my life's pain—family issues, relationship problems, existential angst—with work. When someone praises me for my work, gives me a promotion or offers me a raise, they dull the pain of my life. No wonder I'm hooked on work!

When I no longer suffocate the pained parts of me that I'm trying to repress, it gives me an opportunity to heal the pain itself, straight at the source.

Fourth: I cultivate a set of spiritual practices—meditation, prayer, mindfulness—that accept my fundamental smallness and helplessness in the face of life's challenges. They allow me to feel like an actor in a much larger play. I walk and work more slowly, read books by mystics and spend more time in nature.

I stop being obsessed with trying to be the next Steve Jobs, and ponder how I can take inspiration from Gandhi's life instead. With this new perspective, I have more compassion for myself and others. Instead of judging myself harshly for not being special enough, I wonder how I can be even more loving, even more patient, even more *ordinary*.

Fifth: I disconnect from people and media that promote separateness. For me to feel special, I used to have to see others as broken. But what if life is not a zero sum game? What if I'm just like everyone else, no matter how amazing I am in business? So, I cut out news coverage, TV shows, books, and movies focused on hatred, narcissism, or self-centeredness. I stop being around complaining friends who feel the world is unfair to them.

I begin to cultivate deep, close relationships with a small handful of people, instead of superficial relationships with many. To be ordinary is to live with my own faults, and to accept others' faults. To be exceptional is to imagine a world without faults, and to seek a fantasy where everyone behaves according to my expectations.

The closer I get to annoying family members, troublesome coworkers and quirky friends, the more I begin to see—in myself—all of their faults. I realize I could be loved not only in spite of my faults, but sometimes *because* of them.

My greatest weakness—my feeling of brokenness—becomes my greatest strength. It affirms my humanity and, ultimately, my true ordinariness. ■

KRISHNAN RANGACHARI is a career coach for software engineers. Visit radicalshifts.com to download his free career success kit.

DevExpress Spreadsheet Control

Excel® Inspired UI Controls for Desktops and the Web

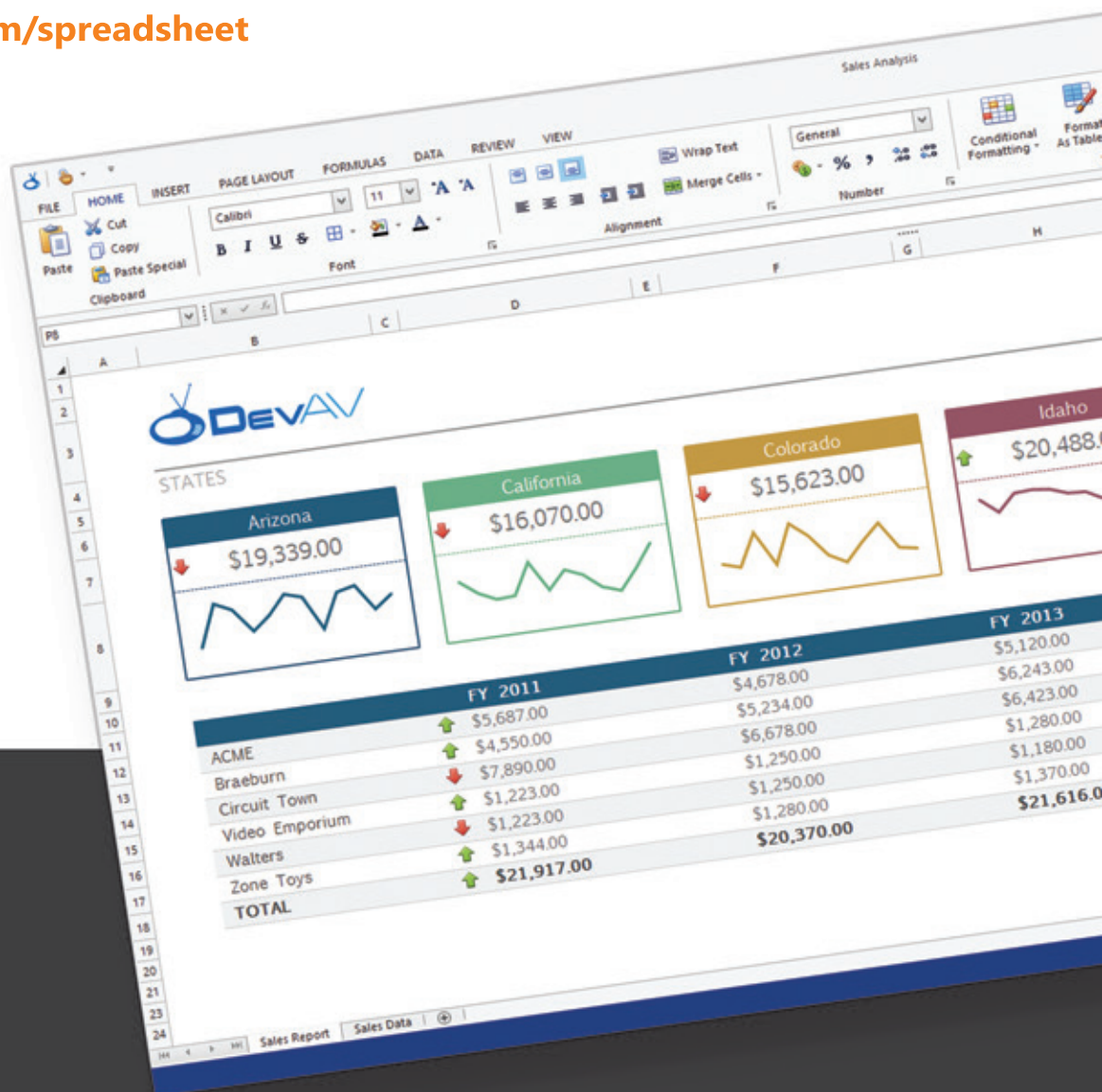
It's no secret...spreadsheets are an important part of business and if you need to incorporate the UX and functionality end-users have come to expect from Microsoft Excel®, DevExpress Spreadsheet is the tool for you. The component library ships with dozens of easy-to-implement features including chart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

WIN WPF ASP MVC

Your Next Great Spreadsheet Starts @DevExpress

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

devexpress.com/spreadsheet





Pushing Notifications to Mobile Apps

In my last few columns, I've written mostly about software design and architecture. While the role of architecture in the building of a software system should never be undervalued or denied, any application in the end results from the sum of individual features. It's the same User Experience Driven Development (UXDD) philosophy that says the measure of success for a software application is in the experience the user goes through when she deals with the application. If your software system comprehends a mobile front end, then you can hardly ignore a feature like push notifications.

In this column, I'll summarize what it takes to add a notification layer on top of mobile applications, regardless of the mobile OS itself. In doing so, I'll review the services of the Microsoft Azure Notification Hub platform.

Push Notifications at a Glance

A push notification is when a mobile device receives a message from the application's back end without first sending an explicit request. Most of the interactions between the client and server components of an application occur through an explicit action—typically, a user action—that solicits feedback. In a way, a push notification is a sort

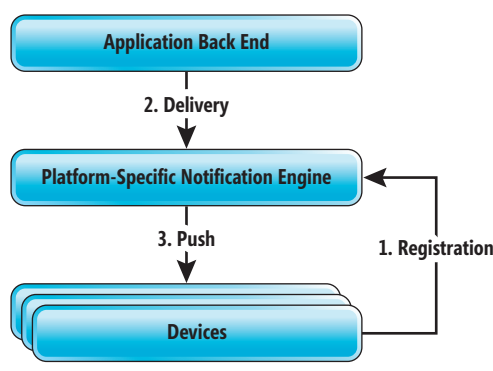


Figure 1 Overall Architecture of a Platform-Specific Push Notification System

The net effect of push notifications on a user is the delivery of relevant updates whether or not he's actively using the app. For example, installing an airliner's app probably enables you to receive quick and timely updates on schedule and gate changes. At some point your phone beeps or vibrates and visible feedback appears somewhere on the phone's interface. Where exactly it appears, however, strictly depends on the version of the OS on which your device is based, as well as app configuration and user settings. Depending on the OS, a push notification can take the form of an icon on the

top bar, a toast message, a badge update and so on.

The crucial point of push notifications is that all of their technical aspects are rigorously platform-specific. For example, the way an application subscribes to a push notification service is different in iOS, Android, Windows Phone and Universal Windows Platform (UWP) apps. Likewise, each platform uses its own payload to deliver messages to connected devices and each platform requires you to configure a different dispatching engine. In spite of significant differences in the actual implementation, I dare say that the overall architecture of a push notification system (PNS) is quite common and looks like the diagram in **Figure 1**.

Working with Multiple Push Notification Systems

It comes as no surprise that the variety of mobile platforms out there also reflects in a variety of push notification platforms. A developer creating a mobile app for multiple platforms must become familiar with multiple different push notification engines and must set up a proper server environment for each. Unless you're developing an app for a single mobile platform with no foreseeable plans to port it to other platforms, you might want to look into cross-platform PNSes, as shown in **Figure 2**. Compared to the diagram in **Figure 1**, the new architecture adds one more layer and offers a single entry point for programming.

The mobile application registers with the generic push notification hub and the application's back end queues messages to the hub. Next, the hub delivers physical messages to the specific mobile platform using the appropriate protocol and payload. By using one of these cross-platform mobile notification systems, you can use a single API to push

The crucial point of push notifications is that all of their technical aspects are rigorously platform-specific.

of unsolicited feedback, a message that the back end sends when some important information is available. To be precise, the term "unsolicited" is not completely accurate here. Any mobile application must subscribe to the available feed in order to receive push notifications. However, once subscribed to the service, notifications arrive in an unsolicited manner.

We didn't invent the Internet...

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



TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

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



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

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

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

connectivity
powered by 

To learn more please visit our website →

www.nsoftware.com

messages to Android, iOS and Windows Phone devices in a single shot.

The Azure Notification Hub is just one of these cross-platform notification systems. Let's see how it works.

The Azure Notification Hub

The first step to use the Azure Notification Hub is setting up an Azure plan. The service is offered in three levels, the lowest of which is free and sets a limit in the number of reachable devices (500) and maximum notifications you can send (1 million). For more information, refer to bit.ly/1Tbimew. To get started, all you need is a free account to create a Notification Hub and a namespace in it. This information comes in handy later to establish the connection from the application's back end to queue messages for the devices.

The most cumbersome part of using push notifications isn't dealing with a cross-platform hub, but meeting the requirements of each mobile platform you want to reach. For example, in order to deliver updates to iOS or Android devices, you must first fully configure your applications with the respective native PNSes. For more information, go to bit.ly/1o15uv2. As you may know, only registered iOS apps can receive push notifications. To register your app with the native Apple Push Notification Service you must first get a certificate from Apple that serves the purpose of uniquely identifying the notifications coming from your app. The same task for Android requires instead an API key you get through the interface of Android Studio. For UWP apps, you must first register with the Windows Store. All platform-specific steps must be accomplished for any platform you intend to support. Any registration information you get must be entered into the Azure Notification Hub, which will be acting on behalf of you with respect to the native PNSes.

It comes as no surprise that the variety of mobile platforms out there also reflects in a variety of push notification platforms.

Registering the Application with the Hub

Let's say you hold a .p12 certificate file and an updated provisioning profile for your iOS application that's enabled to receive notifications from the Apple system. You would be mostly done if you weren't using the Azure Notification Hub. Should you really use an intermediate hub system?

The point is that any platform-specific PNS still leaves a lot of work for the developer to perform commonly requested tasks such as plain broadcasting to all connected devices or just to specific groups of users

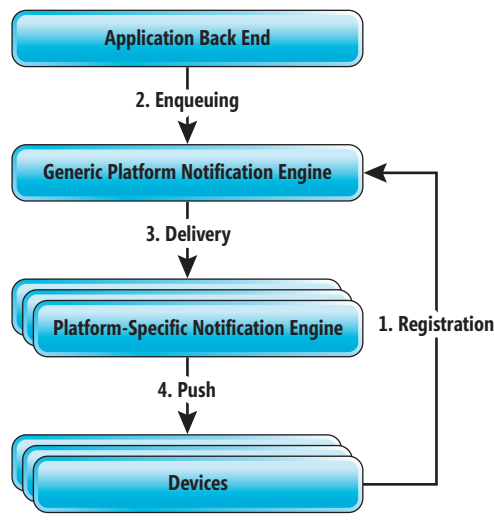


Figure 2 Overall Architecture of a Cross-Platform Push Notification System

such as those using the device in a particular locale. Broadcasting, in particular, is not a trivial task as it may pose nontrivial scalability issues when the number of devices grows. That's why a scalable infrastructure in the middle that decouples core platform-specific services from the code you write is a tremendous benefit. To use the Azure Notification Hub, though, you also need to upload the Apple .p12 certificate and programmatically register your application with the Azure Hub. If you're using Xamarin.iOS to write the iOS app, an excellent step-by-step tutorial is at bit.ly/1KaySJ3.

The mobile application has two main responsibilities. First, it needs to incorporate code that subscribes to the Azure notification feed. Second, it needs to include code to process in some

way any incoming notifications. An iOS app physically receives push notifications from the Apple service so it needs to subscribe to it. This is accomplished through the `UIApplication.SharedApplication.RegisterForRemoteNotifications` method. Upon exit, this method invokes an overridable method in the app delegate class named `RegisteredForRemoteNotifications`. Here, you subscribe to the Azure Hub. This method receives the device token from the OS and your code just forwards it to the Hub. The Azure Hub is identified by path and connection string, like so:

```
var hub = new SBNotificationHub(connectionString, hubPath);
hub.RegisterNativeAsync(deviceToken, null);
```

Next, when a notification is actually received, another overridable method in the app delegate class is invoked: `ReceivedRemoteNotification`. The method receives from the OS the actual content of the pushed message in the form of a (possibly nested) string dictionary. The method override is responsible for extracting the actual message and displaying it through whatever works for the app—badge, sound or alert.

Queuing Messages to the Azure Hub

All the work done so far is only half the effort. What remains is figuring out how to send messages to the Azure Hub and from there down to connected devices. In other words, you need to have an application back end that knows about Hub connection details and passes the message for the user. Such an application back end can be any sort of .NET application, including an ASP.NET application. If you have a business reason for a mobile app to receive push notifications, then something in the business domain is generating related messages. It can be a software trigger to send a message or it can be the action of an admin user, as shown in **Figure 3**.

To incorporate push notifications in an ASP.NET back end, all you need is the Microsoft Azure Notification Hubs NuGet package. In addition, your code is responsible for constructing the proper connection string. A connection string contains information about the related Azure Service Bus URL endpoint and an encrypted

token. The Service Bus endpoint contains the name of the namespace you created when you set up the Azure service. It's something like `sb://your-ns.servicebus.windows.net`. The encrypted token is read from the configuration page of your namespace under the label "Connection String." Here's the code you need to create a valid Hub instance:

```
var hub = NotificationHubClient.CreateClientFromConnectionString(
    connString, hubName);
```

The next step consists of creating the proper payload for each of the platforms you want to target. The payload is a string of JSON that follows a fixed pattern. You can build the JSON string any way you like. In the following example, the `$` is a placeholder for the actual message to send:

```
const string iosFormat = "{\"aps\":{\"alert\":\"$\"}}";
const string androidFormat = "{\"data\":{\"message\":\"$\"}}";
var iosAlert = iosFormat.Replace("$", actualMessage);
var androidAlert = androidFormat.Replace("$", actualMessage);
```

Once the payload is constructed, sending it to the hub is as easy as the following code:

```
var task1 = hub.SendAppleNativeNotificationAsync(iosAlert);
var outcome1 = task1.Result;
var task2 = hub.SendGcmNativeNotificationAsync(androidAlert);
var outcome2 = task2.Result;
```

The outcome variables in the code snippet are instances of the `NotificationOutcome` type and return detailed information about the result of the operation.

Sending Template-Based Messages

The previous code sample shows just the simplest way of sending push notifications—a plain string of text that will be broadcast unchanged to any connected devices. Moreover, you need to format it for each mobile platform of interest. A much more common scenario is sending template-based messages. A template-based message is delivered to Azure in the form of a dictionary of string and the Azure Hub ensures it gets delivered to any mobile platform the account has configured. The key idea behind template-based messages is that the application intends to use a richer format than the default. For example, let's see how to send different messages to users following different locales:

```
var locale = "EN";
var template = String.Format("{\"aps\":{\"alert\":\"$(News_{0})\"}}", locale);
```

The example shows the template to register with the Apple PNS any incoming template-based message named `News_XX` where `XX` is the two initial letters of a locale. The nice thing about templates here is that the application back end might send multiple entries in a single dictionary, but each device receives only the message for which it registered. This is just one of the additional services

brought by an intermediate hub such as the Azure Notification Hub. To send locale-specific messages you need the following code:

```
var messages = new Dictionary<string, string>
{
    {"News_EN", "..."},
    {"News_ES", "..."},
    {"News_PT", "..."},
    {"News_IT", "..."}
};
var task = hub.SendTemplateNotificationAsync(messages);
var outcome = task.Result;
```

With a single push, you reach devices across platforms with the guarantee that each user sees just the notification appropriate for the locale he has selected on the phone. Note that this is a feature slightly different from automatic localization of messages that's available, for example, on iOS devices. On iOS devices, in fact, you can send messages with a placeholder that maps to entries in the localized strings dictionary and the OS does the magic of automatically translating the message before calling the alert. Template messages, instead, are an Azure feature that lets you send different messages to segmented groups of users and you decide how to segment the groups of users.

Scheduled Messages

Another interesting feature you'll find in the Azure Notification Hub is scheduled messages. Scheduled messages are notifications delivered to Azure but sent to connected devices only at a given time. To send scheduled notifications you only use a slightly different API:

```
var notification = new TemplateNotification(messages);
var task = hub.ScheduleNotificationAsync(notification, new DateTime(...));
var outcome = task.Result;
```

It's worth noting that scheduled notifications require a Standard Tier subscription and aren't available in the free test subscription.

Be a Good Citizen

Beyond the mere technical aspects of how to register and send push notifications via the Azure Hub, the real painful point of push notifications is using them in the context of a successful communication with the user.

You don't want to bug the user with plain informational notifications day and night. Because it's a "push" notification, you want to make sure there's a real interest from the user in being pushed. In this regard, segmented groups of users are a great feature on which to rely. The size of the message does matter, too. I suggest you keep any message close to the size of a tweet. Until iOS 8, for example, the maximum size of push notifications was 256 bytes and rose to 2K in newer systems. It's 4K on Android. Last, but not least, make sure the OS you're targeting makes the entire feature easy to opt out. That's mostly true with most recent OSes, but you'd better double check. ■

Figure 3 An ASP.NET Back End to Send Locale-Specific Messages to an iOS and Android App Via the Azure Hub

DINO ESPOSITO is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Modern Web Applications with ASP.NET" (Microsoft Press, 2016). A technical evangelist for the .NET and Android platforms at JetBrains, and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents@wordpress.com and on Twitter: @despos.

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



BEST FILE APIs

Open Create Convert Print Save

files from your *applications!*



XLS

DOC



PDF

Contact Us:

US: +1 888 277 6734

EU: +44 141 416 1112

AU: +61 2 8003 5926

sales@aspose.com

SCAN FOR
20% SAVINGS!



BUSINESS FILE FORMATS



ASPOSE.Cells

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

ASPOSE.Words

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

ASPOSE.Pdf

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

ASPOSE.Slides

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

ASPOSE.BarCode

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

ASPOSE.Tasks

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

ASPOSE.Email

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

ASPOSE.Imaging

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

+ MANY MORE!

Get your FREE evaluation copy at www.aspose.com

.NET

Java

Cloud



Handling the State of Disconnected Entities in EF

Disconnected data is an old problem that precedes Entity Framework and, for that matter, most data access tools. It's never been an easy one to solve. The server sends data down the wire, not knowing what may happen to it in the client app that requested it, not even knowing if it will return. Then, suddenly, some data reappears in a request. But is it the same data? What was it up to in its absence? Did anything happen to it? Is it completely new data? So much to worry about!

As .NET developers, you've likely seen patterns for solving this problem. Remember ADO.NET DataSets? Not only did they contain your data, but they encapsulated all of the change state information for each row and each column. This wasn't limited to "it was modified" or "it is new"; the original data was kept, as well. When we started building ASMX Web services, it was so easy to serialize a dataset and send it down the wire. If that message went to a .NET client, that client could deserialize the dataset and continue to keep track of changes. When it was time to return the data to the service, you would just serialize it again and then, on the server side, deserialize it back into a dataset with all of that lovely change-tracking information intact to be easily persisted to the database. It worked. It was so easy. But it entailed such enormous amounts of data going back and forth across the wire. Not just the data bits, but the structure of the dataset getting serialized created big fat XML.

The size of the serialized message going back and forth across the wire was only one problem. The beauty of Web services was that you could provide services to a variety of platforms, but the message itself was meaningful only to another .NET application. In 2005, Scott Hanselman wrote a great wake-up call to the problem, epically titled, "Returning DataSets from WebServices Is the Spawn of Satan and Represents All That Is Truly Evil in the World" (bit.ly/1TlqcB8).

All of that state information on the wire disappeared when Entity Framework replaced DataSets as the primary data access tool in .NET. Rather than being stored with the data, change-tracking information—original value, current value, state—was stored by EF as part of theObjectContext. But still, in the first iteration of EF, a serialized entity was a cumbersome message due to its need to inherit from the EF EntityObject type. But the message going back and forth across the wire with the entity data had lost its understanding of state. Those of us who were used to the overloaded

DataSet freaked out. Those who were already familiar with handling disconnected state were upset for another reason—the EntityObject base class requirement. Eventually that problem won the EF team's attention (a very good turn of events) and with the next iteration, EF4, EF had evolved to Plain Old CLR Object (POCO) support. This meant that theObjectContext could maintain the state of a simple class with no need for that class to inherit from EntityObject.

But with EF4, the disconnected state problem did not go away. EF had no clue about the state of an entity it was not able to track. People familiar with DataSets expected EF to provide the same magical solution and were unhappy about having to choose between a lightweight message and disconnected change tracking. In the meantime, developers (including me) had explored a lot of ways to inform a server about what happened to the data while it was on its walkabout. You could re-read the data from the database and let EF do a comparison to work out what had changed, if anything. You could make presumptions such as "if the identity key value is 0, it must be new." You could troll around in the low-level APIs to write code to make discoveries about state and act upon them. I did a lot of that back in the day, but none of those solutions were satisfying.

When we started building ASMX Web services, it was so easy to serialize a dataset and send it down the wire.

When EF4.1 came out with its lighter-weight DbContext, it had a gift from the EF team—the ability to easily inform the context about the state of the entity. With a class that inherits from DbContext, you can write code such as:

```
myContext.Entity(someEntity).State=EntityState.Modified;
```

When someEntity is new to the context, this forces the context to begin tracking the entity and, at the same time, specify its state. That's enough for EF to know what type of SQL command to compose upon SaveChanges. In the preceding example, it would result in an UPDATE command. Entry().State doesn't help with the problem of knowing the state when some data comes over the wire, but it does allow you to implement a nice pattern that's now

Code download available at msdn.com/magazine/0416magcode.

in wide use by developers using Entity Framework, which I'll lay out further along in this article.

Even though the next version of Entity Framework—EF Core (the framework formerly known as EF7) will bring more consistency for working with disconnected graphs, the pattern you'll learn in this article should still be useful in your bag of tricks.

The problem with disconnected data escalates as graphs of data get passed back and forth. One of the biggest problems is when those graphs contain objects of mixed state—with the server having no default way of detecting the varying states of entities it has received. If you use `DbSet.Add`, the entities will all get marked `Added` by default. If you use `DbSet.Attach`, they'll be marked `Unchanged`. This is the case even if any of the data originated from the database and has a key property populated. EF follows the instructions, that is, `Add` or `Attach`. EF Core will give us an `Update` method, which will follow the same behavior as `Add`, `Attach` and `Delete`, but mark the entities as `Modified`. One exception to be aware of is that if the `DbContext` is already tracking an entity, it won't overwrite the entity's known state. But with a disconnected app, I wouldn't expect the context to be tracking anything prior to connecting the data returned from a client.

Testing the Default Behavior

Let's clarify the default behavior in order to highlight the problem. To demonstrate, I've got a simple model (available in the download) with a few related classes: `Ninja`, `NinjaEquipment` and `Clan`. A `Ninja` can have a collection of `NinjaEquipment` and be associated with a single `Clan`. The test that follows involves a graph with a new `Ninja` and a pre-existing, un-edited `Clan`. Keep in mind that I would normally assign a value to `Ninja.ClanId` to avoid confusion with reference data. In fact, setting foreign keys rather than navigation properties is a practice that can help you avoid a lot of problems due to the “magic” of EF working out state across relationships. (See my April 2013 column [bit.ly/20XVxQi], “Why Does Entity Framework Reinsert Existing Objects into My Database?” to learn more about that.) But I'm writing the code this way to demonstrate the behavior of EF. Notice that the clan object has its key property, `Id`, populated to indicate that it's pre-existing data that came from the database:

```
[TestMethod]
public void EFDoesNotComprehendMixedStatesWhenAddingUntrackedGraph() {
    var ninja = new Ninja();
    ninja.Clan = new Clan { Id = 1 };
    using (var context = new NinjaContext()) {
        context.Ninjas.Add(ninja);
        var entries = context.ChangeTracker.Entries();
        OutputState(entries);
        Assert.IsFalse(entries.Any(e => e.State != EntityState.Added));
    }
}
```

My `OutputState` method iterates through `DbEntityEntry` objects where the context retains the state information for each tracked entity and prints out the type and value of its `State`.

In the test, I emulate the scenario that, somewhere, I've created a new `Ninja` and associated it with the existing `Clan`. The clan is simply reference data and has not been edited. Then I create a new context and use the `DbSet.Add` method to tell EF to track this graph. I assert that none of the entities being tracked are anything but `Added`. When the test passes, it proves the context didn't

comprehend that the `Clan` was `Unchanged`. The test output tells me that EF thinks both entities are `Added`:

```
Result StandardOutput:
Debug Trace:
EF6WebAPI.Models.Ninja:Added
EF6WebAPI.Models.Clan:Added
```

As a result, calling `SaveChanges` will insert both the `Ninja` and the `Clan`, resulting in a duplicate of the `Clan`. If I had used the `DbSet.Attach` method instead, both entities would be marked `Unchanged` and `SaveChanges` wouldn't insert the new `Ninja` into the database, causing real problems with the data persistence.

Another common scenario is retrieving a `Ninja` and its `Equipment` from the database and passing them to a `Client`. The `Client` then edits one of the pieces of equipment and adds a new one. The true state of the entities is that the `Ninja` is `Unchanged`, one piece of `Equipment` is `Modified` and another is `Added`. Neither `DbSet.Add` nor `DbSet.Attach` will comprehend the varying states without some help. So now it's time to apply some help.

Informing EF of Each Entity's State

The simple recipe for helping EF comprehend the correct state of each entity in a graph consists of a four-part solution:

1. Define an enum representing possible object states.
2. Create an interface with an `ObjectState` property defined by the enum.
3. Implement the interface in the domain entities.
4. Override the `DbContext SaveChanges` to read object state and inform EF.

EF has an `EntityState` enum with the enumerators `Unchanged`, `Added`, `Modified` and `Deleted`. I'll create another enum to be used by my domain classes. This one mimics those four states, but has no ties to the Entity Framework APIs:

```
public enum ObjectState
{
    Unchanged,
    Added,
    Modified,
    Deleted
}
```

`Unchanged` is first so it will be the default. If you want to specify the values, be sure that `Unchanged` is equal to zero (0).

Next, I'll create an interface to expose a property to track the state of objects using this enum. You might prefer to create a base class or add this to a base class you're already using:

```
public interface IObjectWithState
{
    ObjectState State { get; set; }
}
```

Figure 1 Ninja Class Implementing `IObjectWithState`

```
public class Ninja : IObjectWithState
{
    public Ninja() {
        EquipmentOwned = new List<NinjaEquipment>();
    }
    public int Id { get; set; }
    public string Name { get; set; }
    public bool ServedInOniwaban { get; set; }
    public Clan Clan { get; set; }
    public int ClanId { get; set; }
    public List<NinjaEquipment> EquipmentOwned { get; set; }
    public ObjectState State { get; set; }
}
```

This State property is for in-memory use only and doesn't need to be persisted to the database. I've updated the NinjaContext to ensure that the property is ignored for any objects that implement it:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder) {
    modelBuilder.Types<IObjectWithState>().Configure(c => c.Ignore(p => p.State));
}
```

With the interface defined, I can implement it in my classes, for example, in the Ninja class shown in **Figure 1**.

With my default ObjectState enum defined as Unchanged, every Ninja will begin Unchanged and anyone coding with the Ninja class will be responsible for setting the State value as needed.

If relying on the client to set state is a problem, another approach, which is influenced by Domain-Driven Design practices, can ensure that the Ninja object is more involved in its behavior

and state. **Figure 2** shows a much more richly defined version of the Ninja class. Note that:

- The Create factory methods both set the State to Added.
- I've hidden the setters of the properties.
- I've created methods to change properties where the State is set to Modified if it isn't a new Ninja (that is, the state isn't already set to Added).

I've modified the NinjaEquipment type to be richer, as well, and you can see that I benefit from that in the AddNew, Transfer and NoLongerExists equipment methods. The modification ensures that the foreign keys pointing back to the Ninja are persisted correctly or, in the case of equipment being destroyed, that it gets deleted completely from the database according to the business rules of this particular domain. Tracking relationship changes when reconnecting graphs to EF is a little trickier, so I like that I can keep tight control over the relationships at the domain level. For example, the ChangeOwner method sets the State to Modified:

```
public NinjaEquipment ChangeOwner(int newNinjaId) {
    NinjaId = newNinjaId;
    State = ObjectState.Modified;
    return this;
}
```

Now, whether the client explicitly sets the state or uses classes like this (or similarly coded classes in the language of the client) on the client side, the objects passed back into the API or service will have their state defined.

Now it's time to leverage that client-side state in the server-side code.

Once I connect the object or object graph to the context, the context will need to read the state of each object. This ConvertState method will take an ObjectState enum and return the matching EntityState enum:

```
public static EntityState ConvertState(ObjectState state) {
    switch (state) {
        case ObjectState.Added:
            return EntityState.Added;
        case ObjectState.Modified:
            return EntityState.Modified;
        case ObjectState.Deleted:
            return EntityState.Deleted;
        default:
            return EntityState.Unchanged;
    }
}
```

Next, I need a method in the NinjaContext class to iterate through the entities—just before EF saves the data—and update the context's understanding of each entity's state according to the State property of the object. That method, shown here, is called FixState:

```
public class NinjaContext : DbContext
{
    public DbSet<Ninja> Ninjas { get; set; }
    public DbSet<Clan> Clans { get; set; }

    public void FixState() {
        foreach (var entry in ChangeTracker.Entries<IObjectWithState>()) {
            IObjectWithState stateInfo = entry.Entity;
            entry.State = DataUtilities.ConvertState(stateInfo.State);
        }
    }
}
```

I considered calling FixState from inside of SaveChanges so it would be totally automated, but there could be side effects in a number of scenarios. For example, if you use your IObjectState entities in a connected application that doesn't bother setting the local state, FixState will always revert entities to Unchanged. It's better to leave

Figure 2 A Smarter Ninja Class

```
public class Ninja : IObjectWithState
{
    public static RichNinja CreateIndependent(string name, bool servedInOniwaban) {
        var ninja = new Ninja(name, servedInOniwaban);
        ninja.State = ObjectState.Added;
        return ninja;
    }
    public static Ninja CreateBoundToClan(string name,
        bool servedInOniwaban, int clanId) {
        var ninja = new Ninja(name, servedInOniwaban);
        ninja.ClanId = clanId;
        ninja.State = ObjectState.Added;
        return ninja;
    }
    public Ninja(string name, bool servedInOniwaban) {
        EquipmentOwned = new List<NinjaEquipment>();
        Name = name;
        ServedInOniwaban = servedInOniwaban;
    }

    // EF needs parameterless ctor for queries
    private Ninja(){}

    public int Id { get; private set; }
    public string Name { get; private set; }
    public bool ServedInOniwaban { get; private set; }
    public Clan Clan { get; private set; }
    public int ClanId { get; private set; }
    public List<NinjaEquipment> EquipmentOwned { get; private set; }
    public ObjectState State { get; set; }

    public void ModifyOniwabanStatus(bool served) {
        ServedInOniwaban = served;
        SetModifiedIfNotAdded();
    }
    private void SetModifiedIfNotAdded() {
        if (State != ObjectState.Added) {
            State = ObjectState.Modified;
        }
    }
    public void SpecifyClan(Clan clan) {
        Clan = clan;
        ClanId = clan.Id;
        SetModifiedIfNotAdded();
    }
    public void SpecifyClan(int id) {
        ClanId = id;
        SetModifiedIfNotAdded();
    }
    public NinjaEquipment AddNewEquipment(string equipmentName) {
        return NinjaEquipment.Create(Id, equipmentName);
    }
    public void TransferEquipmentFromAnotherNinja(NinjaEquipment equipment) {
        equipment.ChangeOwner(this.Id);
    }
    public void EquipmentNoLongerExists(NinjaEquipment equipment) {
        equipment.State = ObjectState.Deleted;
    }
}
```


Unleash the **UI Superhero** in You

With DevExpress tools, you'll deliver amazing user-experiences for Windows®, the Web and your Mobile world.



UI CONTROLS FOR

DESKTOP / WEB / MOBILE

Unleash the UI Superhero in you and deliver elegant solutions that fully address customer needs today & leverage your existing knowledge to build next generation touch-enabled solutions for tomorrow. Whether it's an Office-inspired app or a data centric analytics dashboard, DevExpress Universal ships with everything you'll need to build your best, without limits or compromise.



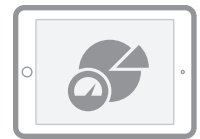
Office-Inspired Apps



Touch-Enabled Windows
& Web Apps



HTML5 Mobile Apps



Reporting-Dashboard
& Analytics Apps

When Only the Best Will Do

With 31 categories and over 400 products to choose from, DevExpress is honored to have been voted best in class 18 times in this year's Visual Studio Magazine Reader's Choice Awards. From the best Component Suite for the Desktop to the best Web Development Toolset, the awards help highlight our commitment to delivering high-performance, easy-to-use and feature-complete tools for Visual Studio developers worldwide.

“ DevExpress has been consistent in its delivery of quality tools and services — and breadth of offerings — year after year. It stands to reason that readers would vote for DevExpress again this year in a number of categories.”

Michael Domingo

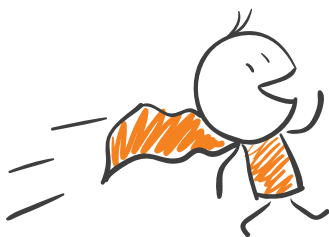
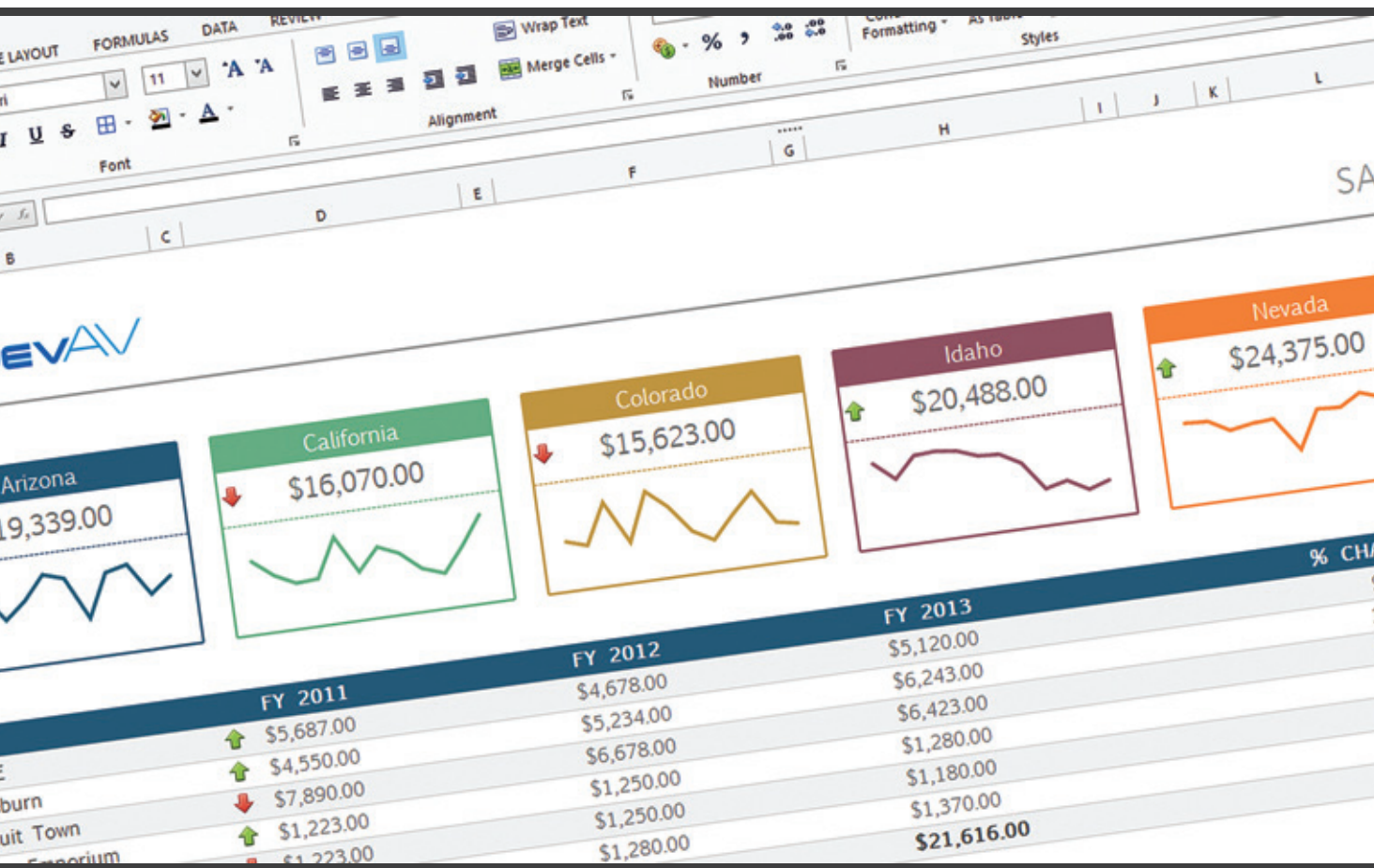
Editor in Chief, Visual Studio Magazine



x13

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

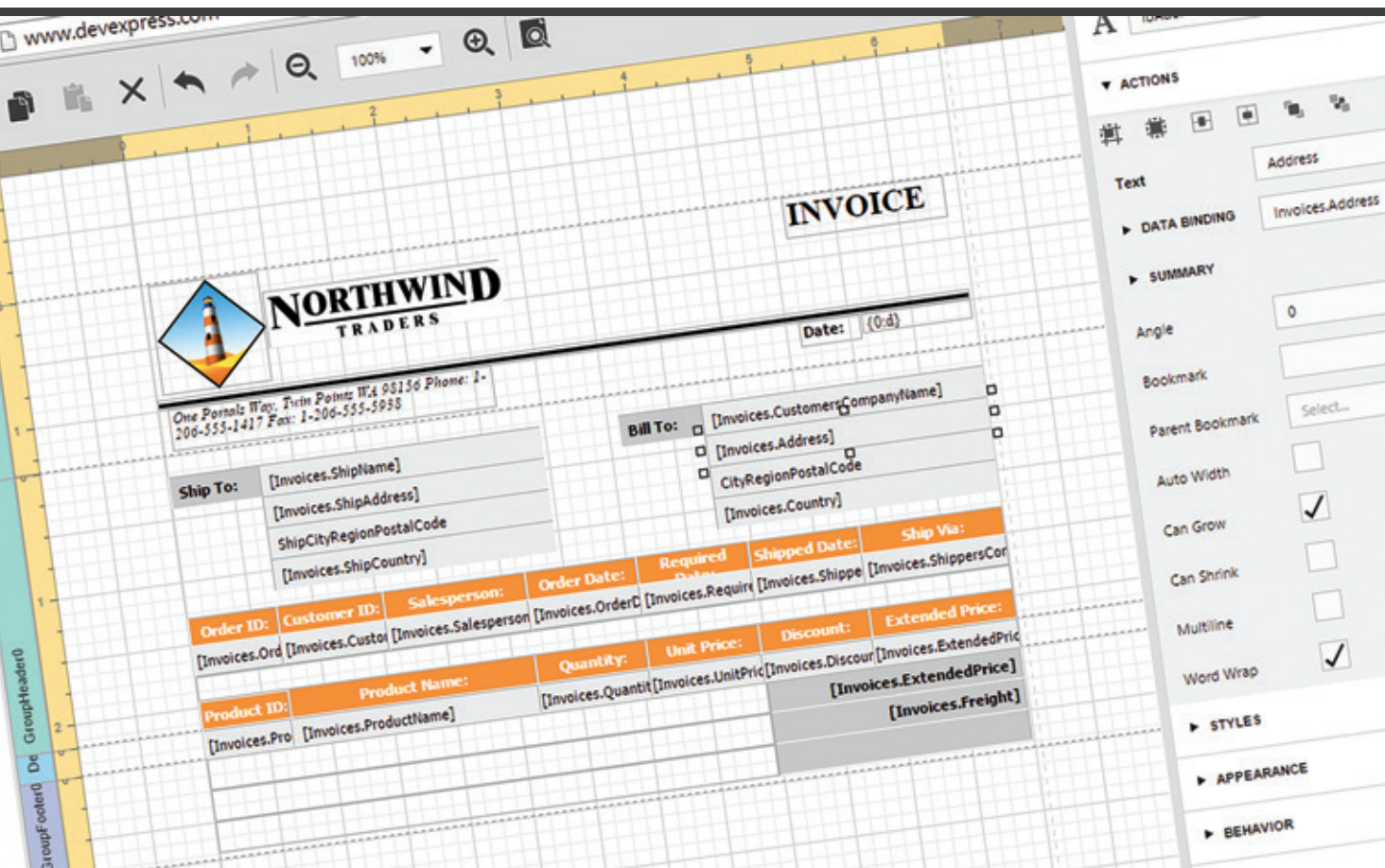
Explore our complete range of Office-inspired controls for all major .NET platforms.

devexpress.com/office



Reporting Made Easy

Inform and analyze with absolute ease. DevExpress Universal includes an enterprise-ready reporting platform with integrated Windows & Web report designers.



Your Next Great Business Report Starts Here

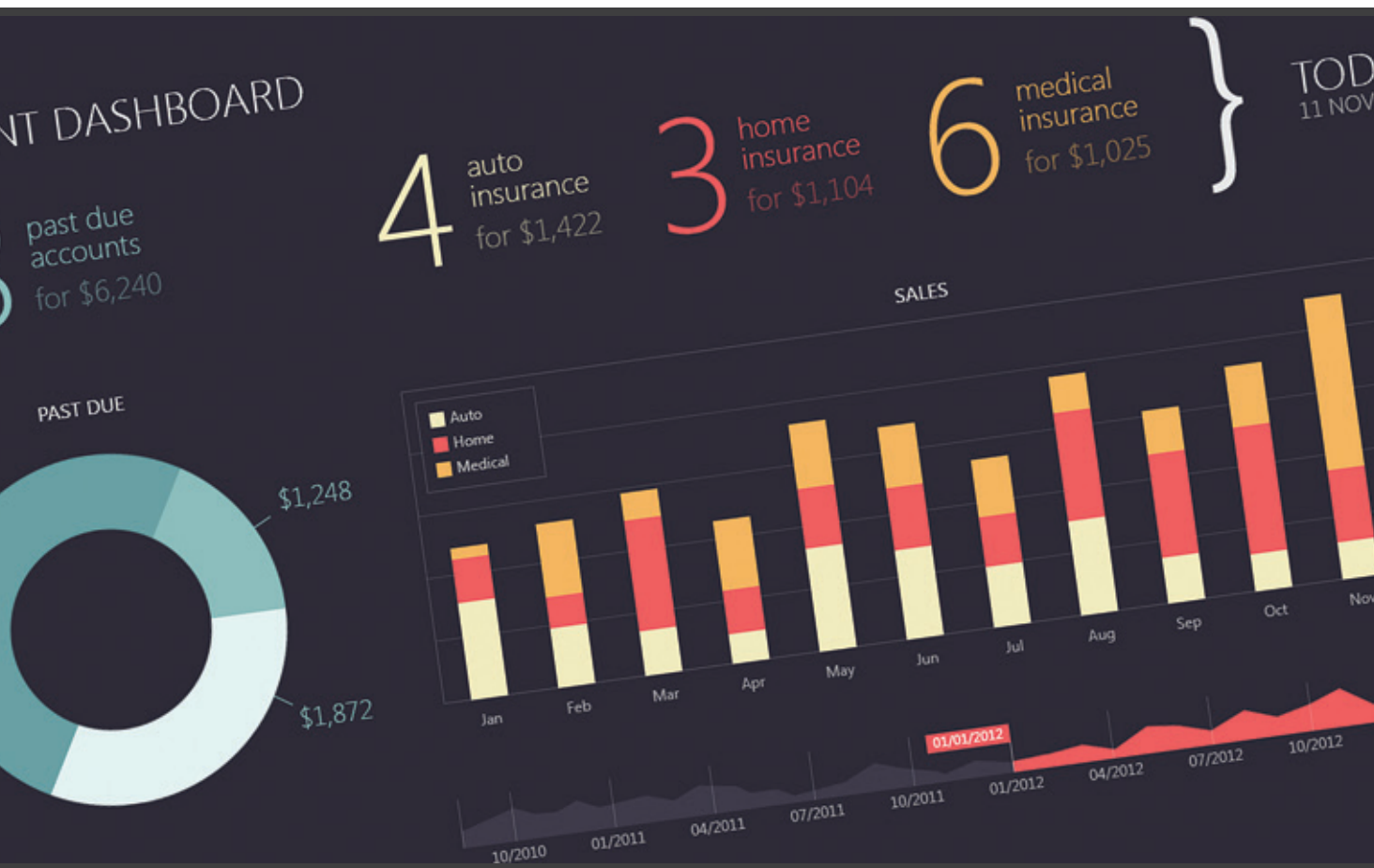
See the power and flexibility of our reporting platform in action.

devexpress.com/reports



Dashboards & Analytics

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



Your Next Great Dashboard Starts Here

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

devexpress.com/dashboard



Touch-Optimized Hybrid Apps

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



Your Next Great Hybrid App Starts Here

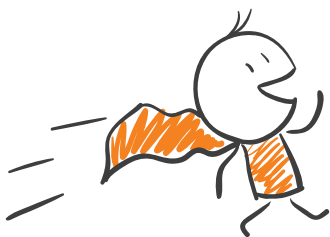
Build touch-first apps for any Windows device and leverage existing code investments.

devexpress.com/hybrid



Mobile Apps for Your BYOD World

Consistent user experiences, everywhere. DevExpress Universal allows you to create apps with a native UX for iOS, Android and Windows Phone, without extra effort, code or UI customization.



Your Next Great Mobile App Starts Here

Put the power of HTML5 and JS into action today.

devexpress.com/mobile



When Only the Best Will Do

Experience the DevExpress difference for yourself
and see why your peers have consistently voted
DevExpress Universal best-in-class.



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



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

it as a method to be executed explicitly. In “Programming Entity Framework: DbContext,” a book I co-authored with Rowan Miller, we discuss some additional edge cases that might be of interest.

Now, I’ll create a new version of the previous test that uses these new features, including the richer versions of my classes in the test. The new test asserts that EF comprehends mixed states for a brand-new Ninja tied to an existing Clan. The test method prints out the EntityState before and after calling NinjaContext.FixState:

```
[TestMethod]
public void EFComprehendsMixedStatesWhenAddingUntrackedGraph() {
    var ninja = Ninja.CreateIndependent("julie", true);
    ninja.SpecifyClan(new Clan { Id = 1, ClanName = "Clan from database" });
    using (var context = new NinjaContext()) {
        context.Ninjas.Add(ninja);
        var entries = context.ChangeTracker.Entries();
        OutputState(entries);
        context.FixState();
        OutputState(entries);
        Assert.IsTrue(entries.Any(e => e.State == EntityState.Unchanged));
    }
}
```

The test passes and the output shows that the FixState method applied the proper state to the Clan. If I were to call SaveChanges, that Clan wouldn’t be reinserted into the database by mistake:

```
Debug Trace:
Before:EF6Model.RichModels.Ninja:Added
Before:EF6Model.RichModels.Clan:Added
After:EF6Model.RichModels.Ninja:Added
After:EF6Model.RichModels.Clan:Unchanged
```

Using this pattern also solves the problem of the Ninja graph I discussed earlier where the Ninja might not have been edited and any number of changes (inserts, modifications or deletes) made to the equipment. **Figure 3** shows a test that checks to see if EF correctly identifies that one of the entries is modified.

The test passes and the output shows that the original Attach method resulted in all objects marked Unchanged. After calling FixState, the Ninja is Unchanged (which is still correct), but the equipment object is correctly set to Modified:

```
Debug Trace:
Before:EF6Model.RichModels.Ninja:Unchanged
Before:EF6Model.RichModels.NinjaEquipment:Unchanged
After:EF6Model.RichModels.Ninja:Added
After:EF6Model.RichModels.NinjaEquipment:Modified
```

What About EF Core?

Even as I move to EF Core, I’ll keep this pattern in my toolbox. Great strides have been made toward simplifying the problems of

Figure 3 Testing State of Children in a Graph

```
[TestMethod]
public void MixedStatesWithExistingParentAndVaryingChildrenIsUnderstood() {
    // Arrange
    var ninja = Ninja.CreateIndependent("julie", true);
    var pNinja = new PrivateObject(ninja);
    pNinja.SetProperty("Id", 1);
    var originalOwnerId = 99;
    var equip = Create(originalOwnerId, "arrow");
    // Act
    ninja.TransferEquipmentFromAnotherNinja(equip);
    using (var context = new NinjaContext()) {
        context.Ninjas.Attach(ninja);
        var entries = context.ChangeTracker.Entries();
        OutputState(entries);
        context.FixState();
        OutputState(entries);
    }
    // Assert
    Assert.IsTrue(entries.Any(e => e.State == EntityState.Modified));
}
```

Self-Tracking Entities

Another feature of EF4.1 was a T4 template that generated “self-tracking entities,” which turned those newly freed PO-COs back into weighed-down beasts. Self-tracking entities were designed specifically for scenarios where Windows Communication Foundation (WCF) services served data to .NET clients. I was never a fan of self-tracking entities and was happy when they quietly disappeared from EF. However, some developers relied on them. And there are some APIs that will give you these benefits. For example, Tony Sneed built a lighter-weight implementation called “trackable entities,” which you can find at trackableentities.github.io. IdeaBlade (ideablade.com) has a rich history of solving disconnected data problems with its flagship product, DevForce, which includes EF support. IdeaBlade took that knowledge and created the free and open source Breeze.js and Breeze# products, which provide client- and server-side state tracking, as well. I’ve written about Breeze previously in this column, in the December 2012 (bit.ly/1WpN0z3) and April 2014 issues (bit.ly/1Ton1Kg).

disconnected graphs—mostly along the lines of providing consistent patterns. In EF Core, setting state using DbContext.Entry().State property will only ever set the state of the root of the graph. This will be advantageous in many scenarios. Additionally, there’s a new method called TrackGraph that will “walk the graph,” hitting every entity within, and apply a specified function to each method. The most obvious function is one that simply sets state:

```
context.ChangeTracker.TrackGraph(Samurai_GK,
    e => e.Entry.State = EntityState.Added);
```

Imagine having that function be one that uses the aforementioned FixState method to apply the EF state based on the ObjectState set on the client side.

Rich Domain Models Simplify Controlling State in the Client

While I prefer building the richer domain classes that update the state as needed, you can achieve the same results with simple CRUD classes as long as the client using the classes explicitly sets the states. With a manual method, however, you’ll have to pay closer attention to modified relationships, ensuring you account for the foreign key modifications.

I’ve been using this pattern for years, and sharing it in books, at conferences, with clients and in my Pluralsight courses. And I know it’s being happily used in many software solutions. Whether you’re using EF5 or EF6, or gearing up for EF Core, this recipe should remove a huge layer of pain related to your disconnected data. ■

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

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

CODE

Anaheim

FOR A BETTER TOMORROW



SEPT. 26-29, 2016

HYATT REGENCY
DISNEYLAND®
GOOD NEIGHBOR HOTEL

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS



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

SUPPORTED BY



Visual Studio
MAGAZINE



PRODUCED BY



ANAHEIM • SEPT. 26-29, 2016

HYATT REGENCY, A DISNEYLAND® GOOD NEIGHBOR HOTEL



CONNECT WITH VISUAL STUDIO LIVE!



twitter.com/vslive – @VSLive



facebook.com – Search “VSLive”



linkedin.com – Join the
“Visual Studio Live” group!

DEVELOPMENT TOPICS INCLUDE:

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

California code with us: register today!

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



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

USE PROMO CODE VSLAN2

VSLIVE.COM/ANAHEIM

.NET Goes Cross-Platform with .NET Core

Phillip Carter and Zlatko Knezevic

At Microsoft, we're building a new implementation of .NET, called .NET Core, to let you write cross-platform code for cloud-optimized workloads. Many are excited about this open source development, but what does it actually mean? This article should help clarify what .NET Core is today and what its goals are, how it relates to the Microsoft .NET Framework and the fundamentals of the command-line tooling that you can use to get started with .NET Core.

What's .NET Core?

To understand what .NET Core is, it's helpful to understand .NET itself. Many people mean ".NET Framework" when they say ".NET," but there's more to it than that. .NET is an ECMA standard that has different implementations—.NET Framework, Mono, Unity and, now, .NET Core. This means that many of the experiences are shared

between the .NET Framework and .NET Core. However, .NET Core is new, with some different principles in mind.

First, .NET Core is cross-platform. It runs on Windows, OS X and multiple distributions of Linux. It also supports different CPU architectures. We're adding more Linux distribution and CPU architecture support with the eventual goal of .NET Core running in as many places as possible.

At the same time, .NET Core is fundamentally modular in its design and architecture. The runtime, library and compiler components are all separate entities that communicate through well-designed interfaces. This allows you to "swap" components in and out for your particular needs. The libraries themselves are also modular and distributed via NuGet, letting you use only what you need so that you can fine-tune the footprint of .NET Core on any given system.

In addition, code written for .NET Core is portable, and can be tuned to run across different supported platforms. Depending on how you target your projects, it's possible to have .NET Core code run on the .NET Framework, Mono and Xamarin platforms, on Windows 8 and Windows Phone, and on the Universal Windows Platform (UWP). To learn more, check out the .NET Platform Standard (bit.ly/10f6W1r).

Finally, .NET Core will be "pay-for-play" and performant. One goal of the .NET Core effort is to make the cost of abstraction clear to developers, by implementing a pay-for-play model that makes obvious the costs that come from employing a higher-level abstraction to solve a problem. Abstractions don't come for free, and that truth should never be hidden from developers. Additionally, .NET Core will favor performance with a standard library that minimizes allocations and the overall memory footprint of your system.

This article discusses:

- .NET Core is a new cross-platform .NET implementation with different goals than the .NET Framework
- You can build ASP.NET Core apps, console apps, libraries/frameworks, and Universal Windows Platform apps on .NET Core today
- The .NET CLI is a set of command line tools you can use to build cross-platform .NET Core assets

Technologies discussed:

.NET Core, .NET Core Command-Line Tools, Microsoft .NET Framework, ASP.NET Core

Scenarios for .NET Core

Today, four scenarios exist in which you can write code for .NET Core: cross-platform ASP.NET Web apps, cross-platform console apps, cross-platform libraries and frameworks, and UWP apps

Built for speed, ASP.NET Core 1.0 is the cross-platform Web stack for .NET Core. If you've ever wanted to do something like deploy your ASP.NET Web app to a container on Linux, you can now do that. To learn more about the breadth of functionality ASP.NET Core offers, check out the documentation at bit.ly/1TqPcIo.

The distinction between
cross-platform libraries and
frameworks can be considered
one of scale.

The scope of cross-platform console apps is actually quite larger than many developers might expect. For example, an ASP.NET Core Web app is, at its core, a console app that reads and writes information to ports—it just so happens to do a lot of other things. A suite of microservices that form the back end of an entire system could each be written as console apps.

The distinction between cross-platform libraries and frameworks is one of scale. Libraries are one of the most natural candidates on which to build things on .NET Core. But at a much larger scale, things like frameworks for distributed computing are good candidates, as well.

Finally, UWP apps that target the family of Windows 10 devices run on .NET Core. You can build a fully featured UWP app that incorporates .NET Core libraries to help build rich Windows 10 apps.

In other words, there are a lot of things you can write for .NET Core today. As the tooling matures and expands, there'll be even more things you can build in the future.

If you have .NET Framework assets that fall into one of these four scenarios, or want to cut your teeth on some new technology, go to bit.ly/1Q1Q18q so you can begin writing some .NET Core code.

How .NET Core Compares with the .NET Framework

The .NET most of you have come to know and love is known as the .NET Framework. So how does .NET Core compare with the .NET Framework? The first thing to keep in mind is that you still use the same languages—C#, F#, Visual Basic—to write all your code. Your experience writing code should look and feel very familiar. However, .NET Core is a new stack—not a subset of .NET Framework. It's best to think of .NET Core and the .NET Framework as two stacks that coincide and co-evolve.

The .NET Framework is and will continue to be the stack to use when writing desktop applications for Windows 7 through Windows 10.

In fact, you can have .NET Framework and .NET Core code live harmoniously together in the same solution. For example, consider a scenario where a .NET Framework GUI (like Windows Forms) consumes services written on .NET Core.

It's helpful to think about the similarities and differences of .NET Core and the .NET Framework from two perspectives: API Surface area and runtime capabilities. **Figure 1** helps illustrate the overlap of APIs between the two platforms.

There are .NET APIs implemented on both .NET Core and the .NET Framework (although sometimes with different underlying implementations). At the same time, .NET Core and .NET Framework both have APIs and capabilities that the other does not. For example, the .NET Framework has multiple GUI frameworks and Windows-specific APIs that aren't present in .NET Core. Likewise, .NET Core has cross-platform capabilities and APIs that the .NET Framework lacks.

Additionally, the .NET Framework is a Windows component that's serviced through Windows Updates. .NET Core employs a completely different model for where it lives and how it's serviced. .NET Core is composed of NuGet packages, with the runtime installed App-Local. This means that applications can “carry” .NET Core with them, enabling them to exist side-by-side with other .NET Core instances on a machine or device. Servicing can then be done per-application and through a package manager, rather than globally through OS updates.

A practical question is this: If you write something on one stack, will it run on the other? Like most answers in life, it depends. If the APIs you use are implemented on both platforms, then you should be able to run your code on .NET Core and the .NET Framework with relatively little work on your part. However, if you assume dependencies on the running environment or use APIs not available on one stack (such as a library for working with XAML-based UIs), your code won't run across both stacks. The .NET Portability Analyzer—available as a command-line tool (bit.ly/1Sd8oIK) or Visual Studio extension (bit.ly/1LqX0aF)—is a tool that will analyze your .dll files and generate a report on how portable your code is from the .NET Framework to .NET Core. We'll be releasing more tools to help with porting in the future.

The Command Line: Your Entry Point for .NET Core

.NET Core comes with a new and revamped foundational toolset that'll be used to develop applications. That toolset is called .NET Core CLI, which is short for .NET Core Command-Line Interface. As with other parts of .NET Core, it's also open source (see [GitHub.com/dotnet/cli](https://github.com/dotnet/cli)) and has a vibrant open source community that's intimately involved with its development.

There are several reasons for introducing a new toolset to the world. First, we have the need to support core development scenarios on all of the platforms that .NET Core supports. Given the diverse set of the platforms, a good command-line experience is a great foundation that we can build on;

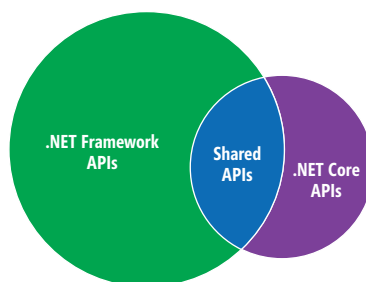


Figure 1 The .NET Framework and .NET Core Share a Subset of APIs

Figure 2 Some Common .NET CLI Commands You Can Use Today

Command	Description
dotnet new	Initialize a valid project for either a class library or a console application using C# as a language.
dotnet restore	Restore dependencies that are defined in the project.json file for a given project. Dependencies are usually NuGet packages you use in your application.
dotnet build	Build your code! This command will produce an Intermediate Language (IL) binary for your project. If the project is a console application, the produced output will be an executable that you can immediately run. By default, the build command will output the built assemblies and executables (if applicable) to the bin directory in the directory where it's invoked.
dotnet test	No good tooling should come without support for running tests. This command allows you to run a suite of tests using a runner that you can specify in the project.json file. Currently supported are xUnit and NUnit test runners.
dotnet publish	Publish your application for running on the targeted machine.
dotnet pack	The pack command will package your project as a NuGet package. The output is a set of nupkg files that you can then upload to your feeds or use in the restore operation by using the local folder override.
dotnet run	The run command will compile and run your application. You can think of this as a Ctrl+F5 analogue, just without Visual Studio.

after all, the command line is what each of these platforms comes with by default.

As a logical extension, we wanted to support the same UX across the supported platforms. You'll be able to move between Linux, Windows, and OS X and not have to re-learn the tools, their syntax, or semantics. They're the same on all platforms. The usage patterns are the same and even the syntaxes are the same.

This idea that there's one toolset you use across the platforms also extends to higher-level tooling, namely Visual Studio Code and Visual Studio. These higher-level tools will be layered upon the .NET CLI and will use them to support .NET Core projects moving forward. This means that when you build your .NET Core application from Visual Studio, .NET CLI tools will be invoked to perform the build.

Trying the .NET Core Command-Line Interface

The easiest way to get started with the .NET Core CLI is to follow the Getting Started guide (aka.ms/dotnetcoregs). The short of it is that you download an installer for your platform (or register a new apt-get feed in the case of Ubuntu), install the tools and you're ready to go. The installers will take care of setting the installation folder on your system PATH in all supported OSes, as well as any other environment variables the CLI needs.

After this, you can start by invoking the driver "dotnet" and passing a command (what we also call a "verb"). The driver is in charge of running the command and passing any arguments to it. At the time of this writing, the CLI package comes with the commands in **Figure 2**. Of course, by the time you read this we'll probably add more commands that will increase your productivity.

In addition to any commands that come with the package, you'll also have an option to add additional commands as tools in your project.json and then restore them. They're packaged as a NuGet package, which provides a nice and easy-to-use and understand extensibility model.

Wrapping Up

We hope you learned about .NET Core and are excited about writing .NET code that can run cross-platform. As a new stack, it provides some exciting capabilities that weren't possible with the .NET Framework before. The .NET CLI also introduces a great command-line experience that will be the foundation of the developer experience and integrated into other tools such as Visual Studio and Visual Studio Code.

Finally, we know that you have a lot of assets written for the .NET Framework, and we'd love to see those assets continue to grow as the .NET Framework evolves. We imagine a world where the .NET

Framework and .NET Core are used together in systems that take advantage of the strengths of both stacks.

If you'd like to learn more and perhaps get involved, here are a few places to get started:

- .NET Core Runtime: [GitHub.com/dotnet/coreclr](https://github.com/dotnet/coreclr)
- The .NET libraries: [GitHub.com/dotnet/corefx](https://github.com/dotnet/corefx)
- The command-line interface and tooling: [GitHub.com/dotnet/cli](https://github.com/dotnet/cli)
- The Roslyn compiler (C# and Visual Basic) and language tooling for Visual Studio: [GitHub.com/dotnet/roslyn](https://github.com/dotnet/roslyn)
- Our documentation for .NET Core: [GitHub.com/dotnet/core-docs](https://github.com/dotnet/core-docs)

We have many more .NET open source projects that are being worked on. If you're interested in seeing more of them, check out the .NET Foundation, an independent organization that fosters open development and collaboration around .NET. Microsoft has contributed several projects to the .NET Foundation, as well as other companies like Xamarin, Umbraco, Salesforce and the .NET community. Learn more about them and contribute at DotNetFoundation.org/projects. ■

PHILLIP CARTER is a program manager on the .NET Team at Microsoft. A lover of systems and programming language theory, Carter can be found staying up late at night arguing over concurrency models with friends. He hopes one day that the runtime behavior of all systems can be expressed in type systems, thereby allowing the human race to ascend to a state of higher reasoning. He can be reached at phcart@microsoft.com.

ZLATKO KNEZEVIC is a program manager (PM) on the .NET Team at Microsoft. In 2005, he joined Microsoft, first working in CEE as a developer evangelist, then moving on to become a PM in SQL Server. There he worked on a range of things, from adding new indices in the core engine to building butt services that deal with Big Data, data discovery and so on. In 2015, he joined the .NET team as a PM and has since been working on .NET Core cross-platform experiences. He can be reached at Zlatko.Knezevic@microsoft.com.

THANKS to the following Microsoft technical experts for reviewing this article: Immo Landwerth and the .NET Core and Framework Program Management Team

Microsoft Cloud Training for the Enterprise

We bring the cloud experts. You choose your pace.

On-Demand Online Training

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

Instructor-Led Training

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

Learning Paths Include:



Microsoft
Azure



Dynamics
CRM Online



Microsoft
Office 365



Visual Studio
Team Services

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

REAL CODE. REAL LABS. REAL LEARNING.

Opsgility.com

 opsgility

Microsoft Pushes C++ into the Future

Kenny Kerr

Visual C++ has a reputation for lagging behind the curve. If you need the latest and greatest C++ features you should just use Clang or GCC, or so the oft-repeated story goes. I would like to suggest that there has been a change to the status quo, a glitch in the matrix, a disturbance in the force, if you will. It's true that the Visual C++ compiler has an incredibly old code base that has made it difficult for the C++ team at Microsoft to add new features rapidly (goo.gl/PjSC7v). This is starting to change, though, with Visual C++ being ground zero for many new proposals to the C++ language and the Standard Library. I'm going to highlight a few new or improved features in the Visual C++ Update 2 release that I've found particularly compelling and that illustrate that there is life yet in this tenured compiler.

Modules

A few developers at Microsoft, notably Gabriel Dos Reis and Jonathan Caves, have been working on a design to add componentization support directly to the C++ language. A secondary goal is to improve build throughput, akin to a precompiled header. This design, called a

module system for C++, has been proposed for C++ 17, and the new Visual C++ compiler provides a proof of concept and the start of a working implementation for modules in C++. Modules are designed to be very straightforward and natural to both create and consume for any developer using standard C++. Make sure you've got Visual C++ Update 2 installed, open a developer command prompt and follow along as I show you how. As the feature is still quite experimental, it lacks any IDE support and the best way to get started is by using the compiler directly from the command prompt.

Let's imagine I have an existing C++ library that I'd like to distribute as a module, perhaps something elaborate, like this:

```
C:\modules> type animals.h
#pragma once
#include <stdio.h>

inline void dog()
{
    printf("woof\n");
}

inline void cat()
{
    printf("meow\n");
}
```

I might also have a compelling sample app to accompany my domesticated library:

```
C:\modules> type app.cpp
#include "animals.h"

int main()
{
    dog();
    cat();
}
```

This article discusses:

- A module system for C++
- Inside coroutines
- Microsoft's investment in C++

Technologies discussed:

Visual C++ Update 2

Pressure from C++ activists has caused me to blush over the use of `printf`, but I can't deny its unmatched performance, so I decide to turn the library into a module to obscure the truth that I prefer `printf` over other forms of I/O. I can start by writing the module interface:

```
C:\modules> type animals.ixx
module animals;
#include "animals.h"
```

I could, of course, just define the cat and dog functions right inside the module interface file, but including them works just as well. The module declaration tells the compiler that what follows is part of the module, but that doesn't mean subsequent declarations are all exported as part of the module's interface. So far, this module doesn't export anything, unless the `stdio.h` header that `animals.h` includes happens to export something all by itself. I can even guard against that by including `stdio.h` prior to the module declaration. So if this module interface doesn't actually declare any public names, how do I go about exporting something for others to consume? I need to use the `export` keyword; this—and the module and import keywords—are the only additions to the C++ language that I need to think about. This speaks to the beautiful simplicity of this new language feature.

The module declaration tells the compiler that what follows is part of the module, but that doesn't mean subsequent declarations are all exported as part of the module's interface.

As a start, I can export the cat and dog functions. This involves updating the `animals.h` header and beginning both declarations with the `export` specifier as follows:

```
C:\modules> type animals.h
#pragma once
#include <stdio.h>

export inline void dog()
{
    printf("woof\n");
}

export inline void cat()
{
    printf("meow\n");
}
```

I can now compile the module interface file using the compiler's experimental module option:

```
C:\modules> cl /c /experimental:module animals.ixx
```

Notice that I also included the `/c` option to instruct the compiler merely to compile but not link the code. At this stage, it wouldn't make sense to have the linker attempt to create an executable. The module option instructs the compiler to produce a file containing metadata describing the interface and the implementation of the module in a binary format. This metadata isn't machine code,

but rather a binary representation for C++ language constructs. However, it's also not source code, which is both good and bad, depending on how you look at it. It's good in that it should improve build throughput because apps that might import the module don't need to parse the code anew. On the other hand, it also means that there isn't necessarily any source code for traditional tools like Visual Studio and its IntelliSense engine to visualize and parse. That means that Visual Studio, and other tools, need to be taught how to mine and visualize code within a module. The good news is that the code or metadata inside a module is stored in an open format and tools can be updated to deal with it.

Moving on, the app can now import the module rather than the library header directly:

```
C:\modules> type app.cpp
import animals;

int main()
{
    dog();
    cat();
}
```

The import declaration instructs the compiler to look for a matching module interface file. It can then use that, along with any other includes that might be present in the app, to resolve the dog and cat functions. Thankfully, the `animals` module exports a pair of furry functions and the app can be recompiled using the same module command-line option:

```
C:\modules> cl /experimental:module app.cpp animals.obj
```

Notice this time that I allow the compiler to call the linker because I now actually want to produce an executable. The experimental module option is still required because the `import` keyword is not yet official. Further, the linker also requires the object file be produced when the module was compiled. This again hints at the fact that the new binary format that contains the module's metadata isn't actually the "code," but merely a description of the exported declarations, functions, classes, templates and so on. At the point at which you want to actually build the app that uses the module, you still need the object file to allow the linker to do its job of assembling the code into an executable. If all went well, I now have an executable I can run just like any other—the end result is no different from the original app using the header-only library. Put another way, a module is not a DLL.

Now, I happen to work on a rather large library, and the thought of adding `export` to every declaration is not at all appealing. Fortunately, the `export` declaration can export more than just functions. One option is to export a bunch of declarations with a pair of braces, as follows:

```
C:\modules> type animals.h
#pragma once
#include <stdio.h>

export
{
    inline void dog()
    {
        printf("woof\n");
    }

    inline void cat()
    {
        printf("meow\n");
    }
}
```


This doesn't introduce a new scope and is merely used to group any contained declarations for export. Of course, no self-respecting C++ programmer would write a library with a bunch of declarations at global scope. Rather, it's far more likely my `animals.h` header declared the dog and cat functions inside a namespace, and the namespace as a whole can be exported quite simply:

```
C:\modules> type animals.h
#pragma once
#include <stdio.h>

export namespace animals
{
    inline void dog()
    {
        printf("woof\n");
    }

    inline void cat()
    {
        printf("meow\n");
    }
}
```

Another subtle benefit of moving from a header-only library to a module is that the app can no longer accidentally take a dependency on `stdio.h` because that's not part of the module's interface. What if my header-only library includes a nested namespace including implementation details not intended for apps to use directly? **Figure 1** shows a typical example of such a library.

Figure 1 Header-Only Library with an Implementation Namespace

```
C:\modules> type animals.h
#pragma once
#include <stdio.h>

namespace animals
{
    namespace impl
    {
        inline void print(char const * const message)
        {
            printf("%s\n", message);
        }
    }

    inline void dog()
    {
        impl::print("woof");
    }

    inline void cat()
    {
        impl::print("meow");
    }
}
```

Figure 2 Preserving the Library's Namespace Structure

```
C:\modules> type animals.h
#pragma once
#include <stdio.h>

namespace animals
{
    namespace impl
    {
        // This is *not* exported -- yay!
    }
}

export namespace animals
{
    // This is exported
}
```

A consumer of this library knows not to take a dependency on anything in the implementation namespace. Of course, the compiler won't stop nefarious developers from doing just that:

```
C:\modules> type app.cpp
#include "animals.h"
using namespace animals;
int main()
{
    dog();
    cat();
    impl::print("rats");
}
```

Can modules help here? Sure, but keep in mind that the design of modules is based upon keeping the feature as small or as simple as possible. So once a declaration is exported, everything is exported unconditionally:

```
C:\modules> type animals.h
#pragma once

#include <stdio.h>

export namespace animals
{
    namespace impl
    {
        // Sadly, this is exported, as well
    }

    // This is exported
}
```

Fortunately, as **Figure 2** shows, you can rearrange the code such that the `animals::impl` namespace is declared separately while preserving the library's namespace structure.

Now all we need is Visual C++ to implement the nested namespace definitions and it becomes quite a bit prettier to look at and a lot easier to manager for libraries with a lot of nested namespaces:

```
C:\modules> type animals.h
#pragma once
#include <stdio.h>

namespace animals::impl
{
    // This is *not* exported -- yay
}

export namespace animals
{
    // This is exported
}
```

Hopefully this feature will arrive in Visual C++ Update 3. Fingers crossed! As it stands, the `animals.h` header will break existing apps that simply include the header and are perhaps built with a compiler that doesn't yet support modules. If you need to support existing library users while slowly transitioning them to modules, you can use the dreaded preprocessor to smooth things over during the transition. This is not ideal. The design of many of the newer C++ language features, including modules, are meant to make programming C++ without macros increasingly plausible. Still, until modules actually land in C++ 17 and commercial implementations are available to developers, I can use a little preprocessor trickery to make the `animals` library build both as a header-only library and as a module. Inside my `animals.h` header, I can conditionally define the `ANIMALS_EXPORT` macro as nothing and use that to precede any namespaces I'd like to export if this were a module (see **Figure 3**).

Now any developer unfamiliar with modules, or lacking an adequate implementation, can simply include the `animals.h` header and use it just like any other header-only library. I can, however, update

Best of the Best: LEADTOOLS Imaging SDKs

Choosing a software development kit for your application can be a difficult task. You must navigate a harmony of features, ease of use, support, cost and more in an equally hard-to-balance time frame. LEADTOOLS is the World Leader in Imaging SDKs and there are many reasons why.

Unparalleled Experience

For over 25 years, LEAD Technologies has been supplying imaging technology to software developers, integrators, contractors and solution providers throughout the United States government, military and most Fortune 1000 companies. LEAD's flagship SDKs, LEADTOOLS, are the most comprehensive and widely used imaging toolkits and are regarded as the gold standard across the globe. Any programmer writing an application for document, medical, multimedia, raster or vector imaging can benefit from the vast development experience packed into the millions of lines of code that comprises LEADTOOLS.

One Stop Shop

If you could summarize LEADTOOLS in one word it would probably be comprehensive. Why? It boasts the most imaging categories on

the widest gamut of development platforms including .NET, Win32/64, WinRT, Windows Phone, HTML5/JavaScript, iOS, OS X, Android, Linux and more. With LEADTOOLS, you will be able to create astonishing applications featuring OCR, Forms, Document Viewing and Conversion, Barcode, PDF, DICOM, PACS, Image Processing, MPEG-2 Transport Stream, Multimedia Playback/Capture/Conversion and much more on virtually any platform you can imagine.

Tomorrow's Technology Today

LEAD Technologies is always at the forefront of imaging technology. This means that you can rest assured that when new standards and development platforms are released that LEADTOOLS will have something for you. LEAD also forms strategic partnerships with customers looking for tools targeting cutting edge technology that provide a dual benefit of a custom-tailored solution and expedited time-to-market.

Personal Touch

We software developers sometimes get a bad rap as being anti-social cave dwellers, but we're people too! LEAD Technologies gets that and has utilized the customer feedback and suggestions over the decades to make LEADTOOLS incredibly programmer-friendly. On top of that, free technical support over email, chat and phone is available and LEAD's developer support team loves to help you get the most out of LEADTOOLS with custom-made example projects and pointers to help you along the way.



the module interface to define `ANIMALS_EXPORT` so that this same header can produce a set of exported declarations, as follows:

```
C:\modules> type animals.ixx
module animals;

#define ANIMALS_EXPORT export

#include "animals.h"
```

Like many C++ developers today, I dislike macros and would rather live in a world without them. Still, this is a useful technique as you transition a library to modules. Best of all, while the app that includes the `animals.h` header will see the benign macro, it won't be visible at all to those who simply import the module. The macro is stripped out prior to the creation of the module's metadata and, as a result, will never bleed into the app or any other libraries and modules that might make use of it.

The type of the coroutine can be deduced by the compiler such that the developer doesn't have to think about what that type might be.

Modules are a welcome addition to C++ and I look forward to a future update to the compiler with full commercial support. For now, you can experiment along with us as we push the C++ standard forward with the prospect of a module system for C++. You can learn more about modules by reading the technical specification (goo.gl/Eyp6EB) or by watching a talk given by Gabriel Dos Reis at CppCon last year (youtu.be/RwdQAOpGWa4).

Coroutines

While coroutines, previously called resumable functions, have been around a little longer in Visual C++, I continue to be excited about the prospect of having true coroutine support in the C++ language—with its deep roots in the stack-based language design of C. As I was thinking what to write, it dawned on me that I wrote not just one but at least four articles about the topic for *MSDN*

Figure 3 Building a Library Both as a Header-Only Library and as a Module

```
C:\modules> type animals.h
#pragma once
#include <stdio.h>

#ifndef ANIMALS_EXPORT
#define ANIMALS_EXPORT
#endif

namespace animals { namespace impl {

// Please don't look here

}}

ANIMALS_EXPORT namespace animals {

// This is all yours

}
```

Magazine. I suggest you start with the latest article in the October 2015 issue (goo.gl/zpPOA0), where you'll be introduced to the coroutines support provided in Visual C++ 2015. Rather than rehash the benefits of coroutines, let's drill into them a little further. One of the challenges with getting coroutines adopted by C++ 17 is that the standardization committee didn't like the idea that they could provide automatic type deduction. The type of the coroutine can be deduced by the compiler such that the developer doesn't have to think about what that type might be:

```
auto get_number()
{
    await better_days {};
    return 123;
}
```

The compiler is more than able to produce a suitable coroutine type and arguably this was inspired by C++ 14, which stated that functions can have their return type deduced:

```
auto get_number()
{
    return 123;
}
```

Still, the standardization committee is not yet comfortable with this idea being extended to coroutines. The problem is that the C++ Standard Library doesn't provide suitable candidates, either. The closest approximation is the clumsy `std::future` with its often heavy implementation and its very impractical design. It also doesn't help much in the way of asynchronous streams produced by coroutines that yield values rather than simply returning a single value asynchronously. So if the compiler can't provide a type and the C++ Standard Library doesn't provide a suitable type, I need to look a little closer to see how this actually works if I'm going to make any progress with coroutines. Imagine I have the following dummy awaitable type:

```
struct await_nothing
{
    bool await_ready() noexcept
    {
        return true;
    }

    void await_suspend(std::experimental::coroutine_handle<>) noexcept
    {}

    void await_resume() noexcept
    {}
};
```

It doesn't do anything, but allows me to construct a coroutine by awaiting on it:

```
coroutine<void> hello_world()
{
    await await_nothing{};
    printf("hello world\n");
}
```

Again, if I can't rely on the compiler automatically deducing the coroutine's return type and I choose not to use `std::future`, then how might I define this coroutine class template?

```
template <typename T>
struct coroutine;
```

Because I'm already running out of space for this article, let's just look at the example of a coroutine returning nothing, or void. Here's the specialization:

```
template <>
struct coroutine<void>
{
};
```

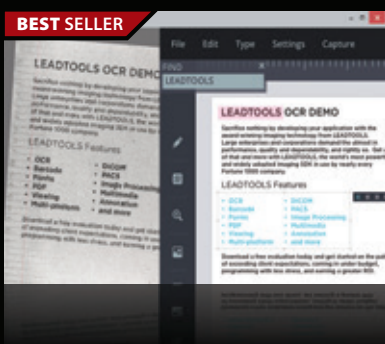


Aspose.Total for .NET | from \$2,449.02



Every Aspose .NET component in one package.

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



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



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

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



DevExpress DXperience 15.2 | from \$1,439.99



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

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



Help & Manual Professional | from \$586.04



Help and documentation for .NET and mobile applications.

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

The first thing the compiler does is look for a `promise_type` on the coroutine's return type. There are other ways to wire this up, particularly if you need to retrofit coroutine support into an existing library, but because I'm writing the coroutine class template I can simply declare it right there:

```
template <>
struct coroutine<void>
{
    struct promise_type
    {
    };
};
```

Next, the compiler will look for a `return_void` function on the coroutine promise, at least for coroutines that don't return a value:

```
struct promise_type
{
    void return_void()
    {}
};
```

While `return_void` doesn't have to do anything, it can be used by different implementations as a signal of the state change that the logical result of the coroutine is ready to be inspected. The compiler also looks for a pair of `initial_suspend` and `final_suspend` functions:

```
struct promise_type
{
    void return_void()
    {}

    bool initial_suspend()
    {
        return false;
    }

    bool final_suspend()
    {
        return true;
    }
};
```

The compiler uses these to inject some initial and final code into the coroutine, which tells the scheduler whether to begin the coroutine in a suspended state and whether to suspend the coroutine prior to completion. This pair of functions can actually return awaitable types so that in effect the compiler could await on both as follows:

```
coroutine<void> hello_world()
{
    coroutine<void>::promise_type & promise = ...;
    await promise.initial_suspend();

    await await_nothing();
    printf("hello world\n");

    await promise.final_suspend();
}
```

Whether or not to await and, thus, inject a suspension point, depends on what you're trying to achieve. In particular, if you need to query the coroutine following its completion, you'll want to ensure that there's a final suspension; otherwise, the coroutine will be destroyed before you get a chance to query for any value captured by the promise.

The next thing the compiler looks for is a way to get the coroutine object from the promise:

```
struct promise_type
{
    // ...

    coroutine<void> get_return_object()
    {
        return ...
    }
};
```

The compiler makes sure the `promise_type` is allocated as part of the coroutine frame. It then needs a way to produce the coroutine's return type from this promise. This then gets returned to the caller. Here I must rely on a very low-level helper class provided by the compiler called a `coroutine_handle` and currently provided in the `std::experimental` namespace. A `coroutine_handle` represents one invocation of a coroutine; thus, I can store this handle as a member of my coroutine class template:

```
template <>
struct coroutine<void>
{
    // ...

    coroutine_handle<promise_type> handle { nullptr };
};
```

Whether or not to await and,
thus, inject a suspension point,
depends on what you're trying
to achieve.

I initialize the handle with a `nullptr` to indicate that the coroutine isn't currently in flight, but I can also add a constructor to explicitly associate a handle with a newly constructed coroutine:

```
explicit coroutine(coroutine_handle<promise_type> coroutine) :
    handle(coroutine)
{
}
```

The coroutine frame is somewhat like a stack frame, but is a dynamically allocated resource and must be destroyed, so I'll naturally use the destructor for that:

```
~coroutine()
{
    if (handle)
    {
        handle.destroy();
    }
}
```

I should also delete the copy operations and allow move semantics, but you get the idea. I can now implement the `promise_type`'s `get_return_object` function to act as a factory for coroutine objects:

```
struct promise_type
{
    // ...

    coroutine<void> get_return_object()
    {
        return coroutine<void>{
            coroutine_handle<promise_type>::from_promise(this);
        }
    };
};
```

I should now have enough for the compiler to produce a coroutine and kick it into life. Here, again, is the coroutine followed by a simple main function:

```
coroutine<void> hello_world()
{
    await await_nothing();
    printf("hello world\n");
}

int main()
{
    hello_world();
}
```

I haven't yet done anything with the result of `hello_world`, yet running this program causes `printf` to be called and the familiar message printed to the console. Does that mean the coroutine actually completed? Well I can ask the coroutine that question:

```
int main()
{
    coroutine<void> routine = hello_world();
    printf("done: %s\n", routine.handle.done() ? "yes" : "no");
}
```

This time I'm not doing anything with the coroutine but asking whether it's done, and sure enough it is:

```
hello world
done: yes
```

Recall that the `promise_type`'s `initial_suspend` function returns `false` so the coroutine itself doesn't begin life suspended. Recall also that `await_nothing`'s `await_ready` function returns `true`, so that doesn't introduce a suspension point, either. The end result is a coroutine that actually completes synchronously because I gave it no reason to do otherwise. The beauty is that the compiler is able to optimize coroutines that behave synchronously and apply all of the same optimizations that make straight-line code so fast. Still, this isn't very exciting, so let's add some suspense, or at least some suspension points. This can be as simple as changing the `await_nothing` type to always suspend, even though it has nothing to do:

```
struct await_nothing
{
    bool await_ready() noexcept
    {
        return false;
    }

    // ...
};
```

In this case, the compiler will see that this awaitable object isn't ready and return to the caller before resuming. Now if I return to my simple hello world app:

```
int main()
{
    hello_world();
}
```

I'll be disappointed to find that this program doesn't print anything. The reason should be obvious: The coroutine suspended prior to calling `printf` and the caller that owns the coroutine object didn't give it an opportunity to resume. Naturally, resuming a coroutine is as simple as calling the handle-provided resume function:

```
int main()
{
    coroutine<void> routine = hello_world();
    routine.handle.resume();
}
```

Now the `hello_world` function again returns without calling `printf`, but the resume function will cause the coroutine to complete. To illustrate further, I can use the handle's `done` method before and after resuming, as follows:

```
int main()
{
    coroutine<void> routine = hello_world();
    printf("done: %s\n", routine.handle.done() ? "yes" : "no");
    routine.handle.resume();
    printf("done: %s\n", routine.handle.done() ? "yes" : "no");
}
```

The results clearly show the interaction between the caller and the coroutine:

```
done: no
hello world
done: yes
```

This could be very handy, particularly in embedded systems that lack sophisticated OS schedulers and threads, as I can write a light-weight cooperative multitasking system quite easily:

```
while (!routine.handle.done())
{
    routine.handle.resume();
    // Do other interesting work ...
}
```

Coroutines aren't magical, nor do they require complex scheduling or synchronization logic to get them to work. Supporting coroutines with return types involves replacing the `promise_type`'s `return_void` function with a `return_value` function that accepts a value and stores it inside the promise. The caller can then retrieve the value when the coroutine completes. Coroutines that yield a stream of values require a similar `yield_value` function on the `promise_type`, but are otherwise essentially the same. The hooks provided by the compiler for coroutines are quite simple yet amazingly flexible. I've only scratched the surface in this short overview, but I hope it's given you an appreciation for this amazing new language feature.

Gor Nishanov, another developer on the C++ team at Microsoft, continues to push coroutines toward eventual standardization. He's even working on adding support for coroutines to the Clang compiler! You can learn more about coroutines by reading the technical specification (goo.gl/9UDeo0) or by watching a talk given by Nishanov at CppCon last year (youtu.be/_fu0gx-xseY). James McNellis also gave a talk on coroutines at Meeting C++ (youtu.be/YytzQ355_Co).

There's so much more happening with C++ at Microsoft. We're adding new C++ language features, including variable templates from C++14 that allow you to define a family of variables (goo.gl/1LbDJ2). Neil MacIntosh is working on new proposals to the C++ Standard Library for bounds-safe views of strings and sequences. You can read up on `span` and `string_span` at goo.gl/zS2Kau and goo.gl/4w6ayn, and there's even an implementation available of both (GitHub.com/Microsoft/GSL).

On the back end, I recently discovered that the C++ optimizer is a lot smarter than I thought when it comes to optimizing away calls to `strlen` and `wcslon` when called with string literals. That's not particularly new, even if it's a well-guarded secret. What is new is that Visual C++ finally implements the complete empty base optimization, which it has lacked for well over a decade. Applying `__declspec(empty_bases)` to a class results in all direct empty base classes being laid out at offset zero. This isn't yet the default because it would require a major version update to the compiler to introduce such a breaking change, and there are still some C++ Standard Library types that assume the old layout. Still, library developers can finally take advantage of this optimization. Modern C++ for the Windows Runtime (moderncpp.com) particularly benefits from this and is actually the reason why this feature was finally added to the compiler. As I mentioned in the December 2015 issue, I recently joined the Windows team at Microsoft to build a new language projection for the Windows Runtime based on moderncpp.com and this is also helping to push C++ forward at Microsoft. Make no mistake, Microsoft is serious about C++. ■

KENNY KERR is a software engineer on the Windows team at Microsoft. He blogs at kennykerr.ca and you can follow him on Twitter: [@kennykerr](https://twitter.com/kennykerr).

THANKS to the following Microsoft technical expert who reviewed this article:
Andrew Pardoe

Using Ionic and TACO to Create Cross-Platform Mobile Apps

Adam Tuliper

There are many workflows available for developing and deploying cross-platform applications, which means it's not always easy to choose. Not only can you get stuck on what development technology to use, but the entire dev-to-production workflow is even more confusing. You can develop natively on each mobile platform, which requires you to be pretty skilled with C#, Objective-C/Swift, Java and sometimes C++. Or you might choose cross-platform tools like Apache Cordova or Xamarin. Beta testing is yet another issue, as you can side-load apps (painful beyond a couple users), use platform-specific beta testing distribution solutions, such as using promotional codes on Windows 10 devices, TestFlight on iOS, groups on Google Play, or platforms like HockeyApp. There are many ways to achieve a similar end result, but through very different journeys.

This article discusses:

- Tools for Apache Cordova (TACO)
- The Ionic framework
- Visual Studio Code
- Live updates to cross-platform Windows Store apps
- Platform-specific CSS

Technologies discussed:

Ionic Framework, Tools for Apache Cordova (TACO), CodePush, Visual Studio and Visual Studio Code

In this article, I'll be taking a look at a workflow for creating cross-platform apps that includes tools installation, development, debugging, deployment (production or beta) and updates. This workflow utilizes the Ionic HTML5 mobile app framework, the Tools for Apache Cordova (TACO) command-line tools, CodePush, and Visual Studio Code, and it can integrate tightly into Visual Studio. I'll focus on the command line and on the Visual Studio Code workflow, as these are essentially the same across Mac OS, Linux, and Windows using the same tools. You'll definitely want the tooling for either Visual Studio Code or Visual Studio—it's the best on the market for developing and debugging Cordova-based apps, though I want to stress the projects are completely interoperable among OSes (Mac OS, Linux, Windows), command lines (Ionic, Cordova and so forth), and tooling (Visual Studio Code or Visual Studio). The tooling for Visual Studio installs all of the required SDKs automatically and sets you up for Windows native and Android debugging.

If you're a Web developer looking to do cross-platform app development and are new to Apache Cordova, I recommend getting started with the article, "Write Cross-Platform Hybrid Apps in Visual Studio with Apache Cordova" (msdn.com/magazine/dn879349), to get an overview of how the framework works.

Tools for Apache Cordova (TACO)

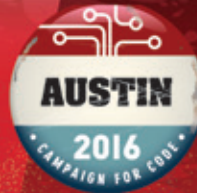
TACO is a suite of products designed to make development faster and more productive. There are multiple endpoints when developing



YOUR CODE COUNTS

CAMPAIGN FOR CODE 2016 • VISUAL STUDIO LIVE!

Austin



PRACTICAL & UNBIASED TRAINING FOR DEVELOPERS:

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

Register by April 13
and Save \$200

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



USE PROMO CODE VSLAPRTI

PLATINUM SPONSOR



GOLD SPONSOR



SUPPORTED BY



PRODUCED BY



VSLIVE.COM/AUSTIN

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/boston

Boston JUNE 13-16
HYATT REGENCY, CAMBRIDGE, MA



PRACTICAL & UNBIASED TRAINING FOR DEVELOPERS:

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

**Register by April 20
and Save \$300**

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



USE PROMO CODE VSLAPRTI



PLATINUM SPONSOR



SUPPORTED BY



Visual Studio
MAGAZINE

PRODUCED BY



VSLIVE.COM/BOSTON

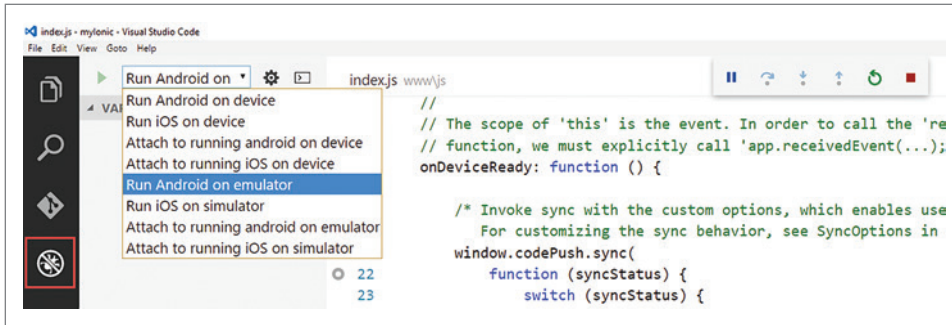


Figure 1 Debugging from Within Visual Studio Code

an Apache Cordova application. The Visual Studio path installs all the required SDK and third-party tools, including an advanced emulator for Android. If you're working on cross-platform development, a useful Visual Studio Code extension gives you features like IntelliSense and debugging from within Visual Studio Code. There's also a set of command-line utilities called TACO-CLI that you can use for installing third-party SDKs if you prefer the command-line path. These tools work with any Cordova-compatible project (Ionic, PhoneGap and more). I'm going to install TACO-CLI now so it's available later when I need it:

```
#install taco one time (**this requires Node to be installed **)
npm install taco-cli -g
```

Whenever you add a platform to an Apache Cordova application via any of the following commands (which basically do the same thing depending on which command line you're using), you're only configuring the project for that platform; the command doesn't do anything about checking for or installing the required SDKs:

```
cordova platform add android
```

```
#or
ionic platform add android
```

```
#or
taco platform add android
```

The Android platform, for example, requires dependencies like the Android SDK, Gradle, and Java, which aren't added by the Ionic or Cordova command-line tools, but can be added from inside an existing project by running:

```
taco install-reqs android
```

This will install Java and the Android SDK, along with Gradle integration and Android Debug Bridge (adb) support.

Developing and Debugging

If you're working cross-platform, Visual Studio Code has tooling that enables you to debug from an emulator or right on a device, as shown in **Figure 1**. All you do is right-click on a folder and select to open with Visual Studio Code, or use File | Open Folder. You can do this with any cross-platform Cordova-based application; there's nothing Visual Studio-specific about this step and it allows you to make the workflow your own, quite a nice feature when you're working with a team using other tools and platforms.

All you need to do to install this extension is to bring up the Command Palette via control-shift-p and type:

```
ext install cordova
```

To debug your app, simply switch to the Debug view (via the icon or Control+Shift+D) and choose your debugging target (as

in **Figure 1**) and click play. This assumes you've already installed the platform-specific SDKs either manually or via taco install-reqs.

If you love Visual Studio, you can also work with existing project folders, such as those from command-line tools, by simply performing File | New | Project From Existing Code. Note that this only adds a .jsproj and .taco.json file in your folder—that's it. You can

then happily use these from Visual Studio while your Mac OS and Linux developers share the same repository and use Visual Studio Code, the command line, and so forth. It all works well together. Of course, you can create a new Apache Cordova project, as well, from File | New Project and that will also work cross-platform. It's a harmonious world, isn't it?

If you're working cross-platform, Visual Studio Code has tooling that enables you to debug from an emulator or right on a device.

What about IntelliSense? In Visual Studio you can install the Ionic Pack from the Extension Gallery right in Visual Studio or from bit.ly/1Vq4dl0. You get snippets, validation and IntelliSense. Visual Studio Code contains IntelliSense for Ionic once you install the Cordova Tools Extension.

The Ionic Framework

Ionic is a CSS framework and a JavaScript UI library, optionally coupled with a build system. It typically runs on top of Apache Cordova, the open source cross-platform HTML/JS/CSS native app framework. You generally hear of Ionic being used with Cordova, but that isn't necessary; for example, check out some of the code pens at CodePen.io/ionic; Ionic is just HTML, CSS, JavaScript and images.

One of the challenges with creating convincing mobile apps is ensuring that they look and perform according to the design guidelines for each OS: the Material Design guidelines for Android; the UX Guidelines for Universal Windows Platform (UWP) apps and the Human Interface Guidelines for iOS. Whether you're developing line-of-business apps or games, users have expectations regarding how an interaction should behave. Whether you realize it, there are micro-interactions you use daily throughout the devices, appliances and other items in your life.

Micro-interactions in a mobile app can be as simple as swiping the screen in a



Figure 2 Android and iOS Spinners

Figure 3 A Native-Looking List Created in Ionic Using HTML

```
<div class="list">
  <a class="item item-icon-left" href="#">
    <i class="icon ion-email"></i>
    Check mail
  </a>
  <a class="item item-icon-left item-icon-right" href="#">
    <i class="icon ion-chatbubble-working"></i>
    Call Ma
    <i class="icon ion-ios-telephone-outline"></i>
  </a>
  <a class="item item-icon-left" href="#">
    <i class="icon ion-mic-a"></i>
    Record album
    <span class="item-note">
      Grammy
    </span>
  </a>
  <a class="item item-icon-left" href="#">
    <i class="icon ion-person-stalker"></i>
    Friends
    <span class="badge badge-assertive">0</span>
  </a>
</div>
```

particular direction to perform an action, pulling to refresh, “liking” on a social media page or displaying a spinner. How do you handle these in a platform-specific manner? Ionic takes care of some of these details automatically. For example, it will detect if you’re on iOS or Android and use the appropriate CSS-styled spinner to look like the platform-specific one, as shown in **Figure 2**. Ionic tags for a spinner are custom HTML tags that look like the following:

```
<ion-spinner></ion-spinner>
```

Figure 3 shows example code for a native-looking list using just HTML and CSS in Ionic. This in turn yields the nice, native-looking top section of the app shown in **Figure 4**.

Getting Started

An Ionic application is just a Web page packaged locally in a Cordova app. There are custom HTML tags as you can see, as well as platform-specific CSS styling. Ionic apps are built on top of Angular. Let’s create a simple Ionic app and explore its structure. If you’re familiar with Angular, this will be a breeze. If not, it’s pretty easy to understand what’s going on from a high level. To use the Ionic command-line interface (cli), you should have npm installed (available when you install Node.js from Nodejs.org) and Git from Git-scm.com.

If you just want to use Ionic in a standalone app, you could simply install it via bower:

```
bower install ionic
```

Visual Studio users can take advantage of the Ionic template (managed by the Visual Studio team) available in the Visual Studio Gallery under Tools | Extensions and Updates. Because Ionic has its own command-line interface (CLI), which makes it pretty easy to use, and it builds on top of Cordova, let’s explore that route and create a new app from the Ionic command line:

```
#install ionic and cordova (once) and create the app(s)
npm install -g ionic cordova
```

```
#create a new app using the tabs template
#Could be one of several templates - run the command:
#ionic templates
#or specify url of a custom template
ionic start myIonic tabs
cd myIonic
```

```
#add platform support to the project
ionic platform add android
ionic platform add ios
```

I don’t have the Android SDK installed so I can’t run it yet. But it’s simple to install it via the TACO-CLI, so I’ll do that and then launch the emulator. Keep in mind that TACO provides much more than just a CLI, as noted earlier. I also want to point out that the Ionic, TACO and Cordova command lines can all work together:

```
#install JAVA, Android SDK, Gradle, and so forth if they don't yet exist
taco install-reqs android
```

```
#run in the browser
ionic serve
```

```
#run in an emulator
ionic emulate android
```

What About iOS?

For iOS, you can debug and run your Apache Cordova application using either Visual Studio Code or Visual Studio. You’ll need access to a Mac at some point if you want to run the app on a device or in the Apple simulator; so if you’re already on Mac OS, you can run and debug directly from within Visual Studio Code as shown previously to launch on a device or in the simulator.

If you prefer to develop on Windows, you have several options. If you’re using Visual Studio, you can use the Apache Ripple simulator on Windows (bit.ly/1Q0qXxK) as it’s configured out of the box to simulate iOS (see **Figure 5**). If you want to run and debug in the Apple simulator (on OS X) or on an iOS device, you’ll need a Mac with Xcode installed and the remote agent installed on the Mac so Visual Studio or Visual Studio Code can communicate with it, as outlined at bit.ly/1XC36H3. You can get a three-month Parallels subscription to run Windows on your Mac from Visual Studio Dev Essentials. Last, you can use one of the various Mac cloud services such as MacInCloud and run in the cloud as shown at bit.ly/1Q0rYpz.

As mentioned, you’ll need Xcode installed on your Mac, but if you use the `taco install-reqs`, that command will also install `ios-sim`, which allows you to launch the iOS simulator from the command line, and use `ios-deploy` to install and debug apps onto iOS devices outside of Xcode.

Loading in Visual Studio Code

At this point, I have an application folder that was created by Ionic. This is a Cordova application. I can load either Visual Studio Code or Visual Studio and point it to this folder and start working with it. I could now go to the Debug

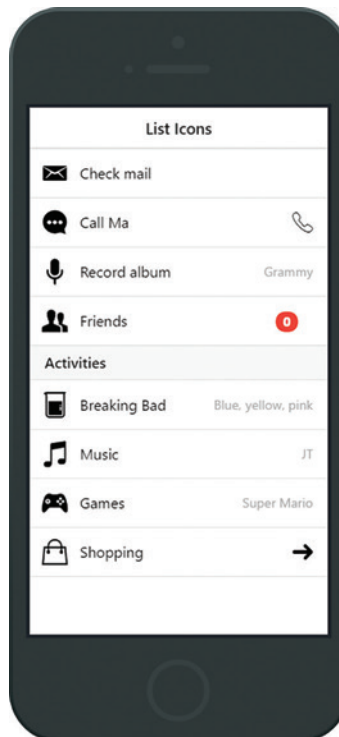


Figure 4 A Native-Looking List Created in Ionic

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

Try a little Vitamin Q.



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

★
**ACT
NOW**
★

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



Solutions for
240+ Countries



10,000+ Customers
Worldwide



30+ Years
Strong



Data Quality &
Mailing Solutions



Cloud • On-Premise
• Services



Germany
www.MelissaData.de

United Kingdom
www.MelissaData.co.uk

India
www.MelissaData.in

Australia
www.MelissaData.com.au

MELISSA DATA®

Your Partner in Global Data Quality

www.MelissaData.com | 1-800-MELISSA

tab in Visual Studio Code, choose a target, and start debugging, as shown in **Figure 1**.

Now let's take a look at the structure of the application. Index.html is the main page and loads Ionic and Angular. You'll quickly note that Ionic uses custom tags, such as `<ion-nav-view>`, but more on those shortly. Index.html is the root page that hosts the other pieces of content delivered through Angular. Ionic consists of two references, the css and the js, which has Angular bundled with it:

```
<!-- compiled css (from sass) output -->
<link href="css/ionic.app.css" rel="stylesheet">
<!-- ionic & angularjs bundled together js -->
<script src="lib/ionic/js/ionic.bundle.js"></script>
```

Figure 6 shows the index.html body tag. The flow in this application is URI-controlled. Those URIs are registered in app.js, as shown in **Figure 7**. If you're an MVC developer, think of these as your routes. **Figure 7** shows just a segment here that'll be extended later. As you can see, two URLs are handled here: /tab and /tab/dash. What gets loaded when the app loads? The `$urlRouterProvider` otherwise function provides a default of /tab/dash, so when you start with index.html, you also process /tab/dash.

The first URI is marked abstract, which means it will never directly navigate to /tabs. It will be utilized, though, whenever any of its children are accessed. Since the default is /tab/dash, this will cause both views to be processed for these URIs. A Cordova app doesn't run in a Web browser but a WebView, so there's no exposed URI a user would type into. This navigation is handled completely via hrefs or JavaScript. The states define what HTML to show for a URI, as well as what controller will handle the business logic.

The flow in this application is URI-controlled.

The `./js/controllers.js` file in this template contains the rudimentary business logic tied to each URI-template combination. The `$scope` variable is used simply to assign data that the templates can use. If I had customer information to return, I could just use:

```
$scope.customer = { "firstName": "John" };
```

Keeping this simplicity in mind, here's the ChatDetail controller assigning data from a Chat service defined in services.js and, in the second example, the controller assigning a JSON object to a variable called settings I can bind in my template:

```
// controller.js
angular.module('starter.controllers', [])
// ...code omitted for brevity
.controller('ChatDetailCtrl', function($scope, $stateParams, Chats) {
  $scope.chat = Chats.get($stateParams.chatId);
})

.controller('StorageCtrl', function($scope) {
  $scope.settings = {
    enableCloudStorage: true
  };
});
```

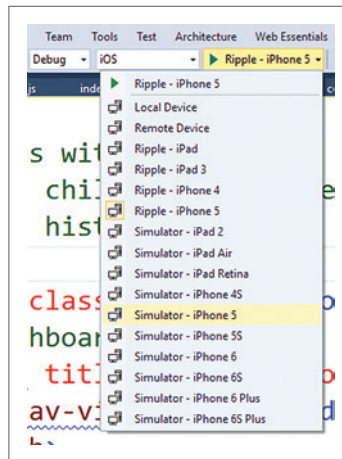


Figure 5 iOS Run/Debug Options in Visual Studio

The default tab interface is made of two templates—the tab interface (from /tab) and whichever of the three child views tabs-html contains, depending what the user selects. The HTML for the interface is shown in **Figure 8**. Notice how I use `ion-tabs` and `ion-tab` to define the tabs control and an individual tab, and then `ion-nav-view` to host the actual child views.

This is the default tab-dash view displayed when the app loads. It's rendered inside of the tab-dash `<ion-nav-view>` noted earlier:

```
<ion-view view-title="Dashboard">
  <ion-content class="padding">
    <h2>Welcome to Ionic</h2>
    <p>
      This is the Ionic starter...
    </p>
  </ion-content>
</ion-view>
```

Ionic also understands navigation between views. If you're running this in a browser and click back, you go to the previous view. If you're running on a Windows Phone or Android device and use the back button, you'll also navigate to the previous view. This is configured inside of Index.html:

```
<ion-nav-bar class="bar-stable">
  <ion-nav-back-button>
</ion-nav-back-button>
</ion-nav-bar>
```

Platform-Specific CSS

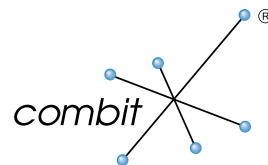
There are still some platform-specific things to do, such as determining which icons to show, but for this I need to know on which platform I'm running. There are several ways to detect this. I could use JavaScript from the default cordova-device plug-in using `device.platform`, but that means being stuck with a bunch of if/else logic to display icons. Remember, these apps are made of Web pages. I don't want to sound simplistic, because "just Web pages" can run full 3D apps at near native speeds (like WebGL), but in this case, because I'm using just HTML/CSS/JS, I can style the app with just CSS. Ionic provides a library called Ionicons, available at [Ionicons.com](http://ionicons.com). And definitely checkout the cheat sheet at ionicons.com/cheatsheet.html.

When you build for a platform by running `cordova build <platform>`, one change that happens is the `<body>` element gets an additional CSS class for that platform. For example, if you run on an Android, the body tag will be some variant of:

```
<body class="platform-android">
```

Figure 6 The Index.Html Body Tag

```
<body ng-app="starter">
  <!--
    The nav bar that will be updated as we navigate between views.
  -->
  <ion-nav-bar class="bar-stable">
    <ion-nav-back-button>
    </ion-nav-back-button>
  </ion-nav-bar>
  <!--
    The views will be rendered in the <ion-nav-view> directive below
    Templates are in the /templates folder (but you could also
    have templates inline in this html file if you'd like).
  -->
  <ion-nav-view></ion-nav-view>
</body>
```



Reporting with List & Label 21

Add value to your application in an instant!

List & Label is the reporting component of choice for thousands of development teams and millions of end-users worldwide – with a successful track record stretching back more than 20 years. It combines high performance with outstanding scalability, and can be integrated into your own application with ease and speed. List & Label gives your users

a powerful reporting tool that enables the import of data from any source – including all SQL dialects, ORM tools, and OData & REST. And it supports a vast number of export and barcode formats, plus Windows 10, the latest .NET features, WinForms, ASP.NET/MVC and WPF apps. The royalty-free designer will add huge value to your applications.



Download your
Free Trial right now.



combit.com/reporting

Figure 7 App.js Containing Angular UI Router

```
// app.js
// ..
stateProvider
// Setup an abstract state for the tabs directive
.state('tab', {
  url: '/tab',
  abstract: true,
  templateUrl: 'templates/tabs.html'
})

// Each tab has its own nav history stack:\
.state('tab.dash', {
  url: '/dash',
  views: {
    'tab-dash': {
      templateUrl: 'templates/tab-dash.html',
      controller: 'DashCtrl'
    }
  }
})
// Extra code removed for brevity...
// If none of the above states (urls)
// are matched, use this as fallback
$urlRouterProvider.otherwise('/tab/dash');
```

You can use this to your advantage to display platform-specific icons by using a mix of the Sass stylesheet language and Ionic Icons. The following command configures my Ionic application with Sass:

```
#tell Ionic to configure/create Sass and make the .scss file
ionic setup sass
```

If you're using Visual Studio, this step works, but be sure to first configure Task Runner Explorer to run the Gulp task to compile the Sass to CSS. Not sure what Gulp or the Task Runner Explorer is? Check out my article on Grunt and Gulp at msdn.com/magazine/mt595751.

If you want Visual Studio to automatically compile your LESS, Sass, CoffeeScript, JSX, ES6 and Stylus files in your Web projects, you can use the Web Compiler extension to process them. Find it at bit.ly/100LP3x. This isn't required here because I'm going to use Gulp, but either method works.

Now let's open up the `./scss/ionic.app.scss` file and add in some Sass. There's loads of information out there about Sass, but the one thing to note here is that it lets you do things with CSS that would otherwise require quite a bit more text to accomplish. If I want a platform-specific cloud icon to show up and a default icon when there isn't a platform-specific one, I can edit the `.scss` file to add the code shown in Figure 9.

Figure 8 The HTML for the Default Tab Interface

```
<ion-tabs class="tabs-icon-top tabs-color-active-positive">
  <!-- Dashboard Tab -->
  <ion-tab title="Status" icon-off="ion-ios-pulse"
    icon-on="ion-ios-pulse-strong"
    href="#/tab/dash">
    <ion-nav-view name="tab-dash"></ion-nav-view>
  </ion-tab>

  <!-- Chats Tab -->
  <ion-tab title="Chats" icon-off="ion-ios-chatboxes-outline"
    icon-on="ion-ios-chatboxes"
    href="#/tab/chats">
    <ion-nav-view name="tab-chats"></ion-nav-view>
  </ion-tab>

  <!-- Storage Tab -->
  <ion-tab title="Storage" icon-off="cloud-outline" icon-on="cloud"
    href="#/tab/storage">
    <ion-nav-view name="tab-storage"></ion-nav-view>
  </ion-tab>
</ion-tabs>
```

Then I can manually trigger the Gulp task for this and run it:

```
#compile sass
gulp sass

Using gulpfile ~\myIonic\gulpfile.js
[11:28:24] Starting 'sass'...
[11:28:26] Finished 'sass' after 2.1 s
#now run it!
ionic emulate android
```

One thing you can immediately see is the platform-specific tab placement, shown in Figure 10. There were no code changes required for this—it's how Ionic operates. Another change, albeit very slight, is the difference in the cloud icon. Because of the simple CSS modification, I'm rendering platform-specific icons for each platform.

Providing Application Updates

Applications are services. In other words, if you want to ensure a healthy community around your applications, you don't deploy once and then forget about them. You should continually update to deliver fixes and features. A very compelling reason to choose Cordova apps over fully native apps is that you can deliver updates out of the store. App approvals on iOS can, for example, take weeks. What do you do if you have a new bug fix or feature that needs to go out immediately?

CodePush is a service from Microsoft that allows out-of-store updates for applications as long as there are no native code changes in the Cordova apps (which would typically be from adding new plug-ins). You can change assets such as the HTML/CSS/JavaScript and create a new release. Releases of your applications are stored in Microsoft Azure and completely managed by CodePush so you don't have to worry about any infrastructure. Let's look at the steps required to bring a version of my Android Ionic app up to the cloud:

```
npm install -g code-push-cli
code-push register
code-push app add myIonic
```

These commands give you a CodePush deployment key, which you can add to the end of the `/config.xml` file using a `<platform>` element:

Figure 9 Using Sass to Define Platform-Specific Cloud Icons

```
// Handle any platform not defined
.cloud{
  @extend .ion-ios-cloud
}

.cloud-outline{
  @extend .ion-ios-cloud-outline
}

// Android-specific
.platform-android {
  .cloud{
    @extend .ion-android-cloud
  }
  .cloud-outline{
    @extend .ion-android-cloud-outline
  }
}

// iOS-specific
.platform-ios .platform-win32{
  .cloud {
    @extend .ion-ios-cloud
  }
  .cloud-outline{
    @extend .ion-ios-cloud-outline
  }
}
```

```
<widget id="com.ionicframework.myionic594688" version="0.0.1"
  xmlns="http://www.w3.org/ns/widgets" xmlns:cdv="http://cordova.apache.org/ns/1.0">
  <!-- other config removed for brevity -->
  <platform name="android">
    <preference name="CodePushDeploymentKey"
      value="YourKeyGeneratedInPriorStep" />
  </platform>
  <platform name="ios">
    <preference name="CodePushDeploymentKey"
      value="YourKeyGeneratedInPriorStep" />
  </platform>
</widget>
```

Now you just need to tell CodePush to check for updates and sync the changes. If changes are found, they're downloaded and updated at the next launch of the app. There are asynchronous and synchronous ways of checking, but I'll keep it short and sweet here with sync. In `app.js`, there's a nice place to check for updates to my application when the Cordova APIs are ready. Typically, in Cordova apps you know you're ready to go by hooking into `deviceready`:

```
window.addEventListener('deviceready', {})
```

Because I'm using Ionic, it will listen for the `deviceready` event and, in turn, call `$ionicPlatform.ready`, which is already provided in the template in `app.js`. I just need to add one line of code there to sync with CodePush:

```
// app.js
angular.module('starter', ['ionic', 'starter.controllers', 'starter.services'])
.run(function($ionicPlatform) {
  $ionicPlatform.ready(function() {
    window.codePush.sync();
    // ...
  });
});
```

I made a small HTML update to the `/templates/tab-dash` file. To release this change, I simply ask Cordova to prepare it, and then CodePush it. First I need the version of the app that's noted in the `/config.xml`:

```
<widget id="com.ionicframework.myionic594688" version="0.0.1"
```

Next, I'll do the whopping two steps to prepare and push this change to the CodePush service:

```
#get my app ready (makes changes only in the /platforms folders)
#we could also cordova prepare or taco prepare - all do the same thing
ionic prepare
```

```
#push this release up to the cloud
#we use /assets as those are the prepared html/js/css files
code-push release myIonic ./platforms/android/assets/www 0.0.1
--deploymentName Production
```

Now, when the application is launched from Visual Studio Code, I can see the app checking for updates in the Visual Studio Code Debug Console:

```
Debug Console
Launching for android (This may take a while)...
App successfully launched
Attaching to android
Forwarding debug port
Attaching to app.
```

```
[CodePush] An update is available.
[CodePush] Downloading update package ...
[CodePush] Package download
[CodePush] Installing update package ...
[CodePush] First update: back up package information skipped.
[CodePush] Install succeeded.
```

That's it! In this configuration, it takes two launches of the app to realize the changes. The first downloads the updates when the new build on CodePush is detected upon startup; the second time the app launches with the applied updates. Note that this entire workflow can be used in production or testing. Everything I've discussed works fine in the emulator, as well as live, and I can see the results in my Android emulator.

Wrapping Up

The workflow I've discussed involves developing, preparing and pushing your apps to the CodePush service. If you're a continuous deployment aficionado, there's a complete workflow outlined at bit.ly/1QpydG4. This workflow additionally outlines deployments from Visual Studio Team System and using HockeyApp to manage beta distributions (crash reporting and user feedback, too!). If you're hoping to hit the ground running, Subhag Oak has a helpful self-contained workshop you can find at bit.ly/1QpCwBt.

Both Angular and Ionic have new major versions in the works (2.x for each of them) that are both available now for checking out. It was a bit early to talk about them here because as of this writing they haven't been released.

This has been a quick tour of one of the most exciting areas of mobile app dev right now. Please check out some of the following resources for great documentation and to keep on top of upcoming changes:

- TACO - taco.tools
 - Visual Studio Tools for Apache Cordova - taco.visualstudio.com
 - Performance tips - bit.ly/24kEalx
 - Native transitions for Ionic Framework - bit.ly/1SKFFfm
 - Visual Studio Tools for Apache Cordova Team Blog - bit.ly/1KBpJcH
- Until next time!

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 channel9.msdn.com/Blogs/AdamsGarage.

THANKS to the following Microsoft technical experts for reviewing this article: Subhag Oak, Ricardo Minguez Pablos and Ryan Salva

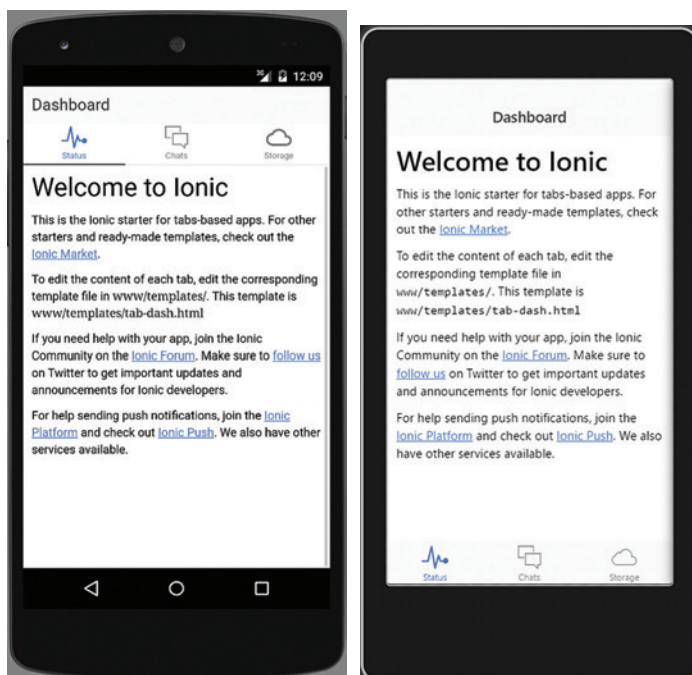


Figure 10 Ionic on Android and iOS

Data Processing and Machine Learning on Spark

Eugene Chuvyrov

Here's a question for you: What's the name of the framework that borrowed heavily from the Microsoft Dryad project, became the most popular open source project of 2015 and also set a data processing record, sorting 100TB of data in just 23 minutes? The answer: Apache Spark.

In this article, I'll talk about the speed and popularity of Spark and why it's the clear current winner in the Big Data processing and analytics space. Using your Microsoft Azure subscription, I'll present examples of solving machine learning (ML) problems with Spark, taking a small step from software engineering into the data science world. But before I dive into data analysis and ML, it's important to say a few words about various components of the Spark framework and about Spark's relationship with Azure.

Spark Components

The value of the Spark framework is that it allows for processing of Big Data workloads on the clusters of commodity machines. Spark

Core is the engine that makes that processing possible, packaging data queries and seamlessly distributing them across the cluster. Besides Spark Core, there are several additional components to the Spark framework, and each of those components is applicable to a specific problem domain. It's possible that you would never need to work with any of those components if you're interested only in manipulating and reporting on large data workloads. However, in this article, I'll use Spark MLlib to build out an ML model that'll let you fairly accurately "guess" the digits that've been written by hand (a lot more on this later). Other components of the Spark framework allow for processing of streaming data (Spark Streaming), manipulation of graphs and the computation of the famous Page-Rank algorithm (GraphX), and running SQL queries on top of distributed data (Spark SQL).

Running Spark on Azure

There are quite a few options in experimenting with Spark, from using managed services from databricks.com (this company created and continues to enhance Spark), to provisioning a Docker container and grabbing pre-installed Spark images from Docker Hub, to getting the entire source code repo from GitHub (github.com/apache/spark) and building the product yourself. But because this article is about Azure, I'd like to show you how to create Spark clusters on Azure. The reason this option is extremely interesting is because Azure provides enterprise-level guarantees for Spark deployed onto Azure compute clusters. Azure gives a 99.9 percent Microsoft-backed SLA for all Spark clusters and also offers 24x7 enterprise support and cluster monitoring. These guarantees, coupled with the ease of cluster deployment and a slew of announcements around Spark and Azure during the 2016 Build conference make Microsoft Cloud an excellent environment for your Big Data jobs.

This article discusses:

- The accessibility of hyperscale data processing in the Microsoft Cloud
- Apache Spark, a modern Big Data analytics framework, can be deployed in minutes on Azure
- Examples of solving machine learning problems with Spark

Technologies discussed:

Microsoft Azure, HDInsight, Apache Spark, Python

Code download available at:

[GitHub.com/echuvyrov/SparkOnAzure](https://github.com/echuvyrov/SparkOnAzure)

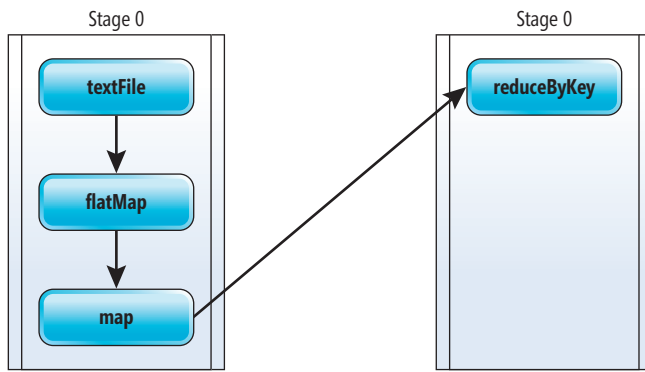


Figure 1 Directed Acyclic Graph (DAG) for Word Count

The Pixie Dust

The secret that makes Spark so popular among data scientists today is two-fold: It's fast and it's a joy to program on that framework. First, let's look at what makes Spark much faster than the frameworks that preceded it.

Spark's predecessor, Hadoop MapReduce, was the workhorse of Big Data analytics space ever since Doug Cutting and Mike Cafarella co-founded the Apache Hadoop project in 2005. MapReduce tools were previously available only inside the Google datacenters and were completely closed sourced. Hadoop worked well for running batch analytics processing on the cluster, but it suffered from extreme rigidity. Map and Reduce operations go together; first you complete the Map task, then you complete the Reduce task. Complex tasks had to combine multiple map and reduce steps. Also, every task had to be decomposed into a map to reduce operations. That took a long time to run these sequential operations and was tedious to program. In other words, this wasn't real-time analytics.

In contrast, the Spark framework applies intelligence to data analytics tasks at hand. It constructs a Directed Acyclic Graph (DAG) of execution before scheduling tasks, very similar to how SQL Server constructs a query execution plan before executing a data retrieval or manipulation operation. DAGs provide information about transformations that'll be performed on data and Spark is able to intelligently combine many of these transformations into a single stage and then execute transformations all at once—an idea initially pioneered by Microsoft Research in Project Dryad.

Additionally, Spark is able to intelligently persist data in memory via the constructs called Resilient Distributed Datasets (RDDs)—which I'll explain later—and share it among DAGs. This sharing of data among DAGs lets jobs complete faster than they would have without that optimization. **Figure 1** shows a DAG for the “hello world” of the data science space—the count of words in a given text file. Notice how several operations, namely reading text file, flatMap and map, are combined into a single stage, allowing for faster execution. The following code shows the actual Scala code (because Spark is written in Scala) performing the word count (even if you've never seen a line of Scala code before, I'm willing to bet that you'll instantly understand how to implement the word count in Spark):

```
val wordCounts = textFile.flatMap(line => line.split(" ").map(word => (word,1))
    .reduceByKey((a, b) => a + b))
```

The second reason Spark is so popular is because of its programming model. Implementing the word count in Spark (Scala code) is much simpler than implementing word count in Hadoop Map-Reduce. In addition to Scala, you can create Spark applications in Java and Python, which is the language I'm using in this article. Before Spark, with Hadoop MapReduce, data scientists/programmers had to use an unnatural paradigm of breaking down a complex task into a set of maps and reduce operations. With Spark, a functional programming approach familiar to any .NET developer working with LINQ and lambda functions is used to transform and analyze data.

Shortly, you'll see just how easy, yet powerful, the Spark programming model is. But before you get to write awesome functional code that'll work equally well on datasets large and small, you need to create a distributed cluster of machines that'll have all the necessary components of Spark installed and ready to accept programming tasks you submit to it. The creation of a Spark cluster would be absolutely daunting if you had to create and configure the cluster yourself; fortunately, Microsoft Cloud lets you accomplish provisioning in just a few clicks. In the next section, I'll show you just how to do that.

Deploying a Spark Cluster

Now, let's create an HDInsight cluster on Azure. Think of “HDInsight” as an umbrella term that includes both Hadoop and Spark technologies; Hadoop HDInsight and Spark HDInsight are two examples of managed Big Data services on Azure.

To provision a Spark cluster, log on to the Azure Portal (portal.azure.com)

and click through New | Data + Analytics | HDInsight | Create. Fill out HDInsight Cluster properties, specifying Name, Cluster Type = Spark, set Cluster Operating System as Linux (because Spark is being developed on Linux) and leave the version field unchanged, as shown in **Figure 2**. Complete the rest of the required information, including specifying the credentials to log onto the cluster and storage account/container name. Then press the Create button. The process of creating a cluster takes 15 to 30 minutes.

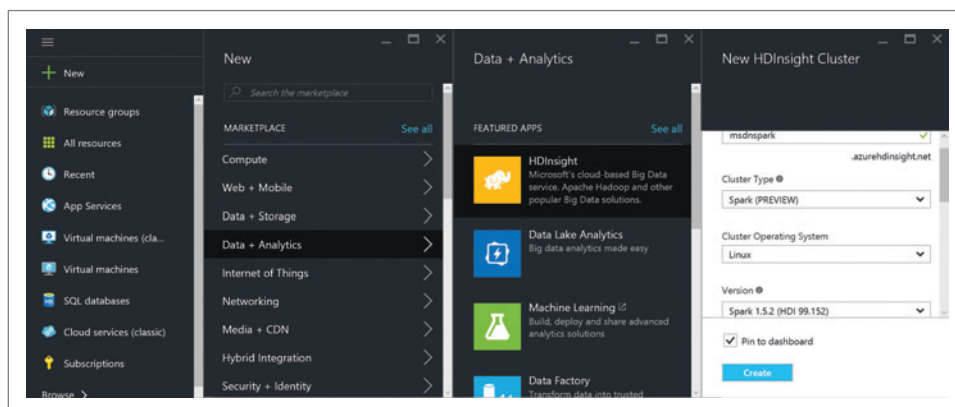


Figure 2 Creating a Spark Cluster in Azure

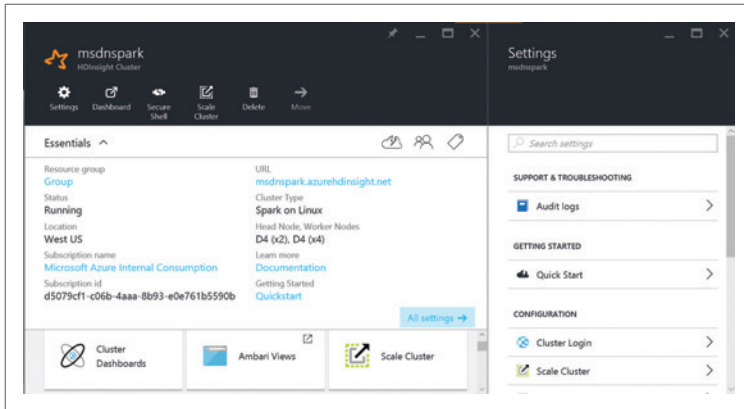


Figure 3 Accessing Jupyter Notebooks in Azure HDInsight Via Cluster Dashboards

After the creation process completes, you'll have a tile in the Azure portal representing the newly created HDInsight cluster. Finally, you get to dive deep into code! Before getting into code, however, let's review the programming environment and languages available to you with Spark.

There are several ways to program in the Spark environment. First, you can access Spark shell via, intuitively enough, the `spark-shell` command, explained at bit.ly/1ON5Vy4, where, after establishing an SSH session to the Spark cluster head node, you can write Scala programs in a REPL-like manner and submit programming constructs one at a time (don't worry if this sentence sounded like it was written in a foreign language, just proceed directly to Option 3). Second, you can run complete Scala applications on Spark (submitting them via the `spark-submit` command explained at bit.ly/1fqgZHY). Finally, there's also an option to use Jupyter notebooks (jupyter.org) on top of Spark. If you aren't familiar with the Jupyter project, Jupyter notebooks provide a visual, Web-based interactive environment in which to run data analytics scripts. These notebooks are my preferred method of data analysis and I'm convinced that, once you try them, they'll become your preferred method of programming on Spark, too. Azure HDInsight installs the Jupyter notebook environment on top of the cluster for you, making it easy to start using it.

To access Jupyter notebooks, click on the Cluster Dashboards tile as illustrated in **Figure 3**, then click the Jupyter notebook tile on the slide-out window. Log in using the credentials you specified during cluster creation and you should see the Jupyter environment ready to



Figure 4 Zoomed in Sample of the Digit "7" Represented in the MNIST Dataset

accept new or edit old notebooks. Now click on the New button in the upper-right corner and select Python 2. Why Python 2? Because while Spark itself is written in Scala and a lot of Spark programming is done in Scala, there's also a Python bridge available via Pyspark. By the way, there's a raging debate whether you should code in Scala or Python. Each language has its clear benefits, with Scala being potentially faster, while

Python is perhaps more expressive and the most commonly used language for data science (see bit.ly/1WTSemP). This lets you use an expressive, yet concise Python when programming on top of the Spark cluster. Python is also my preferred language for data analysis (along with R) and I can utilize all the powerful Python libraries that I'm used to.

You're finally ready to dive deep and perform ML and data analytics tasks inside the Jupyter notebooks.

Machine Learning with Spark

To illustrate ML in Spark, I'll use a "smallish" data example in the form of classical problems in ML—recognizing handwritten digits, such as the ones that appear in ZIP codes on envelopes. Although this dataset isn't large by any means, the beauty of this solution is that, should the data increase one-thousand fold, you could add more

machines to the cluster and still complete the data analysis in a reasonable amount of time. No changes to the code illustrated here will be necessary—the Spark framework will take care of distributing workloads to individual machines in the cluster. The data file that you'll use is also a classical one—it's frequently referred to as MNIST dataset—and it contains 50,000 handwritten digits, ready for you to analyze. Although there are many places online to get the MNIST dataset, the Kaggle Web site gives you convenient access to that data (see bit.ly/1QJN20c).

As a side note, if you're not familiar with kaggle.com, it hosts ML competitions online, where almost 500,000 data scientists from around the world compete for monetary prizes or a chance to interview at one of the top ML companies. I've competed in five Kaggle competitions and, if you're a competitive person, it's an extremely addictive experience. And the Kaggle site itself is running on Azure!

Let's take a moment to understand the contents of `train.csv`. Each line of that file represents a pixel-by-pixel representation of a 28x28 image containing a handwritten digit, such as the one shown in **Figure 4** (the figure shows a zoomed-in representation). The first column contains what the digit really is; the rest of the columns contain pixel intensities, from 0 to 255, of all 784 pixels (28x28).

With the new Jupyter notebook open, paste the following code into the first cell:

```
from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import RandomForest
import time

sc = SparkContext(appName="MNISTDigitsDT")
#TODO: provide your own path to the train.csv in the line(s) below, you can use
Azure Storage #Explorer to upload files into the cloud and to read their full path
fileNameTrain = 'wasb://datasets@chuvyrov.blob.core.windows.net/trainingsample.csv'
fileNameTest = 'wasb://datasets@chuvyrov.blob.core.windows.net/validationsample.csv'

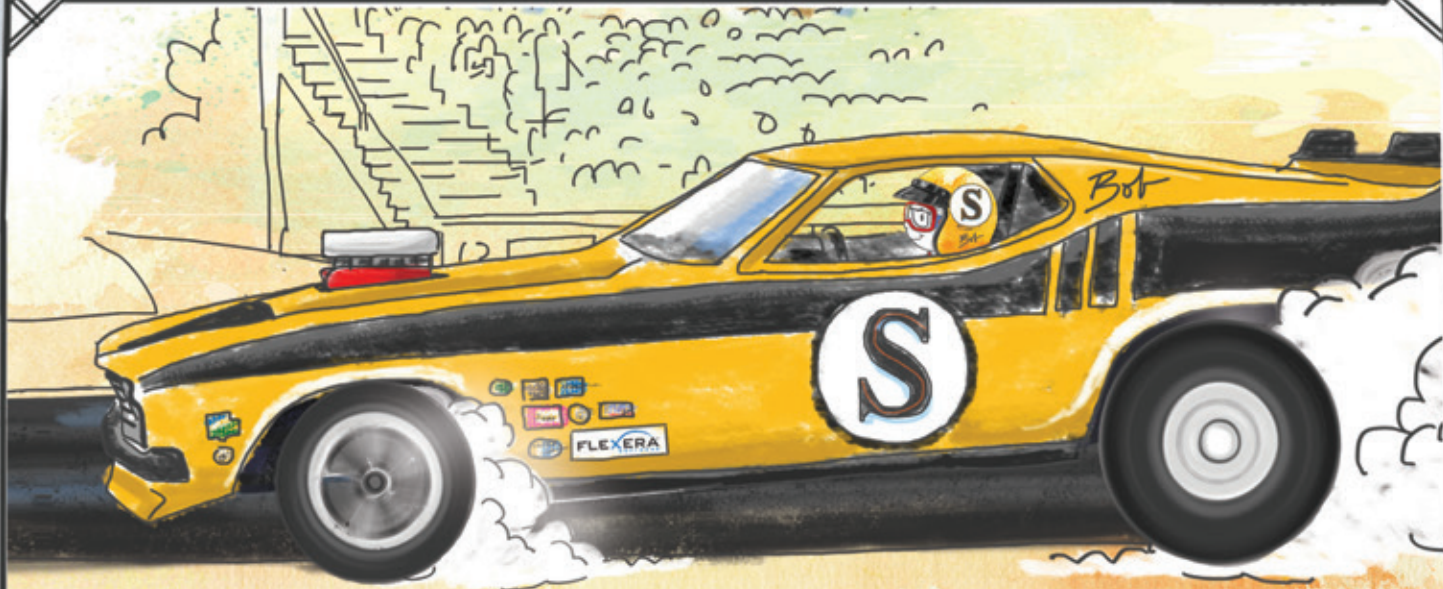
mnist_train = sc.textFile(fileNameTrain)
mnist_test = sc.textFile(fileNameTest)
```

This code imports the necessary libraries for doing ML in Spark, then specifies the location of the data files that'll be used for training and testing the model (note that these files should reside in your storage account, accessible from Spark in Microsoft Cloud via the `wasb://` reference). Finally, the last two lines is where the RDD are created from the text files. RDDs are the magic behind Spark—they're distributed data structures, but the complexity of



FLEXERA SOFTWARE®

InstallShield® GO FOR THE WIN



VISIT FLEXERA SOFTWARE AT MICROSOFT BUILD 2016

Take a Fresh Look at InstallShield

Inside Track – Simplify complex installs for web, cloud, virtual, PC and server

Wow the Crowd – Reliably deliver fast downloads and simple installations

Pole Position – Spend less time building installations and more time creating features that put you in the lead

Team Win – Distributed and agile teams can easily collaborate and save time

*Keep competitors in your rearview mirror.
Test drive InstallShield today.*



Watch the Video:

flexerasoftware.com/GoForTheWin



FLEXERA
SOFTWARE

US/Canada: 1-800-809-5659
International: 1-847-466-4000

Ask to speak to an InstallShield Account Manager

their implementation is generally hidden from the programmer/user. Additionally, these RDDs are lazily evaluated and are persisted, so in case you needed to use that RDD again, it's immediately available without re-computation/retrieval. When you manipulate RDDs, it triggers the generation of DAGs and the execution of staged tasks in the Spark cluster, as I touched upon earlier.

Press Shift+Enter inside the Jupyter cell to execute the code you pasted. No news should be good news (if you didn't get an error message, you're good), and you should now have RDDs available to you for querying and manipulation. These RDDs contain lines of comma-separated text at the moment, because that's how your MNIST data came through.

The next thing you're going to do is define a simple function that'll help you convert these lines of text into a custom LabeledPoint object. This object is required for the ML algorithm that you'll use to train and make predictions. In a nutshell, this object contains an array of "features" (sometimes, it's convenient to think of features as columns in a database table) or characteristics about a single data point, as well as its "label," or the value you're trying to learn to predict. If this sounds a bit unclear right now, perhaps looking at the MNIST train.csv file might help. You'll notice that every line in train.csv has a number in the first column and a set of numbers, from 0 to 255, in all other columns. The first column is called the "label," because we're trying to learn how to predict that number. All other columns are "features" and the features taken all together are referred to as a "feature vector." These features are the intensity of each digitized pixel in the picture of the digit, 0 being black and 255 being white, with many values in between. The pictures are all 28 pixels high and 28 pixels wide, making up 784 columns containing pixel intensities in the train.csv file (28x28=784).

Copy and paste the following function into the new cell of your Jupyter notebook:

```
def parsePoint(line):
    #Parse a line of text into an MLlib LabeledPoint object
    values = line.split(',')
    values = [0 if e == '' else int(e) for e in values]
    return LabeledPoint(int(values[0]), values[1:])
```

Press Shift+Enter to execute the code. You have now defined the parsePoint function, which has been evaluated by Spark and it's available for you to use with the dataset that you just read in. This function takes in a single line of comma-separated text, splits it into individual values and converts these values into the LabeledPoint object.

Next, you perform some basic data cleansing to get it ready for the learning algorithm; unfortunately, the learning algorithm isn't yet smart enough to know what part of the data has predictive value. So, skip the header of the train.csv file using a hack borrowed from stackoverflow.com; then, you'll print the first line of the resulting RDD to make sure it's in the state you expect it to be in:

```
#skip header
header = mnist_train.first() #extract header
mnist_train = mnist_train.filter(lambda x: x != header) #filter out header
using a lambda
print mnist_train.first()
```

Now, you're ready to apply a functional programming approach with the .map(parsePoint) operator in the next section to transform the RDD into the format ready for ML algorithms in Spark. This transformation will essentially parse every line inside the mnist_train RDD and convert that RDD to a set of LabeledPoint objects.

RDDs and Interactivity: Main Pillars of Spark's Power

There are several important issues here. First, you're working with a data structure distributed across the cluster of machines (the RDD), yet the complexity of distributed computing is almost completely hidden from you. You're applying functional transforms to the RDD, and Spark optimizes all the processing and heavy lifting across the cluster of available machines behind the scenes for you:

```
labeledPoints = mnist_train.map(parsePoint)
#Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = labeledPoints.randomSplit([0.7, 0.3])

print mnist_train.first()
```

Although the last line (with the print statement) might seem trivial, the ability to interactively query large datasets is extremely powerful and was virtually absent from the world of large datasets before Spark. In your data science and large data manipulation projects, it'll be a very useful technique to verify that the transformations you think are being applied are indeed being applied. This powerful interactive processing is yet another advantage of Spark over other Big Data processing frameworks.

Also notice the splitting of the data into the training and test dataset using the randomSplit function. The idea there is to create an ML model using the data in trainingData RDD and to test the model using the data in testData RDD, as you'll see in the code shortly.

You're now ready to apply an ML algorithm to the distributed dataset that you've just created (mnist_train). As a quick review, remember that in ML problems, almost in all cases there are two distinct sets of steps that occur: First, you train the model using the known dataset with known conclusions; second, you make predictions based on the model you created, or learned, in the first step. In the following code, you're using a RandomForest algorithm available within the Spark Machine Learning framework (Spark MLlib) to train the model. RandomForest is one of several distributed algorithms available within Spark MLlib and it's one of the most powerful. Paste the following contents into the new cell:

```
depthLevel = 4
treeLevel = 3
#start timer
start_time = time.time()
#this is building a model using the Random Forest algorithm from Spark MLlib
model = RandomForest.trainClassifier(trainingData, numClasses=10,
    categoricalFeaturesInfo={},
    numTrees=treeLevel, featureSubsetStrategy="auto",
    impurity='gini', maxDepth=depthLevel, maxBins=32)

print("Training time --- %s seconds ---" % (time.time() - start_time))
```

Figure 5 Evaluating the Accuracy of Your Predictions

```
# Evaluate model on test instances and compute test error

1 #start timer
2 start_time = time.time()
3 #make predictions using the Machine Learning created prior
4 predictions = model.predict(testData.map(lambda x: x.features))
5 #validate predictions using the training set
6 labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
7 testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() /
8 float(testData.count())
9 print('Test Error = ' + str(testErr))
10 print("Prediction time --- %s seconds ---" % (time.time() - start_time))

11 #print('Learned classification tree model:')
12 #print(model.toDebugString())
```



APPS

READY FOR THE CLOUD!



With the **1&1 Cloud App Center** you can get your applications up and running in no time. Choose from more than **100 cutting-edge apps** and combine them with the superior speed and performance of the **best value cloud server around**.

- ✓ **Secure and powerful platform**
- ✓ **No prior server knowledge necessary**
- ✓ **Billing by-the-minute**



☎ 1 (877) 461-2631



1and1.com

*1&1 Cloud Server is free for one month, after which the regular pricing starts from \$4.99/month. One-time setup fee of \$9.99 applies. 1&1 Cloud Server comes with a 30-day money back guarantee. If your package is not canceled within the first 30 days, you will be charged in accordance with your agreed billing cycle. Visit 1and1.com for full offer details, terms and conditions. Intel, the Intel Logo are trademarks of Intel Corporation in the U.S. and/or other countries. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are property of their respective owners. ©2016 1&1 Internet. All rights reserved.

Note how this code starts to measure the execution time of the algorithms, then sets initial values for some of the parameters expected by the RandomForest algorithm, namely `maxDepth` and `numTrees`. Execute that code by pressing Shift+Enter. You might be wondering what this RandomForest thing is and how does it work? RandomForest is an ML algorithm that, at a very high level, works by constructing *many* decision trees on the data by randomly selecting a variable to split a decision tree on (that is, one tree could be as simple as, “If the pixel in the bottom-right corner is white, it’s probably No. 2”) and then making the final decision after polling all the trees constructed. Fortunately, there’s already a distributed version of the algorithm available to you on Spark. However, nothing stops you from writing your own algorithms should you decide to do it; distributed k-Nearest Neighbors (kNN) algorithm still isn’t present in the Spark framework.

Now, back to the MNIST digits recognition task. If you have an environment similar to mine, you should get execution time of training the algorithm of about 21 seconds. This means that in 21 seconds, you’ve learned—using the RandomForest algorithm—a model that you can use to predict the digits you’re seeing given the features you’ve analyzed. Now you’re ready for the most important part of the ML task—making predictions based on the model you’ve created. In addition, you’re also ready to evaluate the accuracy of these predictions, as shown in Figure 5.

Note the `model.predict` construct on line 4 of Figure 5. This is the line that makes the actual prediction based on the model that you’ve built earlier. On the lines after the prediction is made (lines 5-7), you’re using some basic data manipulation strategies to temporarily relate—via the `zip` function—your predicted values to the real values available to you as part of the download. Then, you sim-

ply compute the percentage of correct predictions given this data and print the execution time.

The result of this initial classification with the error being so high is slightly disconcerting (that is, does your model work at all with error rates approaching 43 percent?). You can improve the model using the concept called “grid hyperparameter search” where you try a series of values when building out the model, test it right away and eventually converge on the hyperparameter values that give you the best performance overall. In other words, try a bunch of systematic experiments to determine what model parameters have the best predictive value.

The hyperparameters that you’ll apply grid search to will be `numTrees` and `maxDepth`; paste the code shown in Figure 6 into the new cell in the notebook.

Notice how in lines 8-14 you scan through a set of `numTrees` parameters for the random forest algorithm from 3 to 10, creating models and evaluating their performance. Next, in lines 30-32, you capture the model if it gives you better results than any of the prior models that you’ve tried, or dismiss the model otherwise. Give this loop some time to run; at the end of the run, you should see prediction error values no greater than 10 percent.

Wrapping Up

When I set out to write this article, my main goal was to show through examples how easy it is to program with Spark, especially if you’re a fan of functional programming and Azure. My secondary goal was to demonstrate how you can perform ML tasks on datasets both large and small with the help of the Spark MLLib library. Along the way, I wanted to explain why Spark performs faster on distributed data than its predecessors and share bits of trivia of how we arrived at where we are today in the distributed data analytics space.

Microsoft is investing heavily in the future of Big Data, ML, analytics and, specifically, Spark. This is the right time to learn these technologies to truly take advantage of the opportunities for hyperscale compute and data analytics provided by Microsoft Cloud. Azure makes getting going with Spark fast, easy and ready to scale up to large datasets, all backed by service-level guarantees you can expect only from the best enterprise cloud providers.

Now that you’ve created an ML model and made predictions on the known data, you can also make predictions on the data that doesn’t include its true label; namely, on the `test.csv` file from Kaggle.com. You can then make a submission to Kaggle.com as part of the digit recognizer competition on that platform. All of the code for this article, as well as the code to write a submission file, is available at [GitHub.com/echuvyrov/SparkOnAzure](https://github.com/echuvyrov/SparkOnAzure). I’d love to learn about the scores that you get. E-mail me with questions, comments, suggestions and ML achievements at eugene.chuvyrov@microsoft.com. ■

Figure 6 Iterative “Grid Search” for Optimal Parameters in the RandomForest Algorithm in Spark

```
1 bestModel = None
2 bestTestErr = 100
3 #Define a range of hyperparameters to try
4 maxDepths = range(4,10)
5 maxTrees = range(3,10)
6
7 #Loop over parameters for depth and tree level(s)
8 for depthLevel in maxDepths:
9     for treeLevel in maxTrees:
10
11         #start timer
12         start_time = time.time()
13         #Train RandomForest machine learning classifier
14         model = RandomForest.trainClassifier(trainingData,
15             numClasses=10, categoricalFeaturesInfo={},
16             numTrees=treeLevel, featureSubsetStrategy="auto",
17             impurity='gini', maxDepth=depthLevel, maxBins=32)
18
19         #Make predictions using the model created above
20         predictions = model.predict(testData.map(lambda x: x.features))
21         #Join predictions with actual values from the data and determine the error rate
22         labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
23         testErr = labelsAndPredictions.filter(lambda (v, p): v != p)
24             .count() / float(testData.count())
25
26         #Print information about the model as we proceed with each iteration of the loop
27         print ('maxDepth = {0:1f}, trees = {1:1f}: trainErr = {2:5f}'
28             .format(depthLevel, treeLevel, testErr))
29         print("Prediction time --- %s seconds ---" % (time.time() - start_time))
30         if (testErr < bestTestErr):
31             bestModel = model
32             bestTestErr = testErr
33
34 print ('Best Test Error: = {0:3f}\n'.format(bestTestErr))
```

EUGENE CHUVYROV is a cloud solutions architect at Microsoft in the Technical Evangelism and Development team where he helps companies around the San Francisco Bay area take full advantage of the hyper scale afforded by Microsoft Cloud. Although he currently focuses on high-scale data partners, he hasn’t forgotten his roots as a software engineer and enjoys writing cloud-ready code in C#, JavaScript and Python. Follow him on Twitter: @EugeneChuvyrov.

THANKS to the following Microsoft technical expert for reviewing this article: Bruno Terkaly

JOIN US on the CAMPAIGN TRAIL in 2016!

	<p>MAY 16 - 19 HYATT AUSTIN, TX vslive.com/austin See pages 48 – 49 for more info</p>	
	<p>JUNE 13 - 16 HYATT CAMBRIDGE, MA vslive.com/boston See pages 50 – 51 for more info</p>	
	<p>AUGUST 8 - 12 MICROSOFT HQ, REDMOND, WA vslive.com/redmond See pages 78 – 79 for more info</p>	
	<p>SEPTEMBER 26 - 29 HYATT ORANGE COUNTY, CA – A DISNEYLAND® GOOD NEIGHBOR HOTEL vslive.com/anaheim See pages 18 – 19 for more info</p>	
	<p>OCTOBER 3 - 6 RENAISSANCE, WASHINGTON, D.C. vslive.com/dc See pages 62 – 63 for more info</p>	
	<p>PART OF LIVE! 360</p> <p>DECEMBER 5 - 9 LOEWS ROYAL PACIFIC ORLANDO, FL live360events.com/orlando</p>	

YOUR CODE COUNTS

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

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



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

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

**REGISTER BY APRIL 13
& SAVE \$200!**

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



USE PROMO CODE VSLAPR5

PLATINUM SPONSOR



GOLD SPONSOR



SUPPORTED BY



PRODUCED BY



AUSTIN AGENDA AT-A-GLANCE

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

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

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

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

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

Speakers and sessions subject to change

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

CONNECT WITH VISUAL STUDIO LIVE!

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

VSLIVE.COM/AUSTIN

BETTER

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

CODE

Boston

FOR ALL

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

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

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

**REGISTER BY APRIL 20
& SAVE \$300!**

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

USE PROMO CODE VSLAPR5



PLATINUM SPONSOR

Microsoft Virtual Academy
LEARNING HAPPENS HERE

SUPPORTED BY

Visual Studio

msdn
magazine

Visual Studio
MAGAZINE



PRODUCED BY

1105MEDIA[®]
YOUR GROWTH. OUR BUSINESS.


BOSTON AGENDA AT-A-GLANCE

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

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

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

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, June 15, 2016			
8:00 AM	9:15 AM	W01 AngularJS & ASP.NET MVC Playing Nice - Miguel Castro	W02 Cloud Enable .NET Client LOB Applications - Robert Green	W03 Creating Great Windows Universal User Experiences - Danny Warren	W04 Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - Nick Landry
9:30 AM	10:45 AM	W05 Did a Dictionary and a Func Just Become the New Black in ASP.NET Development? - Chris Klug	W06 Azure Mobile Apps: APIs in the Cloud for Your Mobile Needs - Danny Warren	W07 User Experience Case Studies—The Good and The Bad - Billy Hollis	W08 Build Cross-Platform Mobile Apps with Ionic, Angular, and Cordova - Brian Noyes
11:00 AM	12:00 PM	General Session: To Be Announced, <i>Tim Huckaby, Founder & Chairman, InterKnowledge & Actus Interactive Software</i>			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch • Visit Exhibitors			
1:30 PM	2:45 PM	W09 Richer MVC Sites with Knockout JS - Miguel Castro	W10 Breaking Down Walls with Modern Identity - Eric D. Boyd	W11 Knockout in 75 Minutes (Or Less...) - Christopher Harrison	W12 Mobile App Development with Xamarin and F# - Rachel Reese
3:00 PM	4:15 PM	W13 Securing Client JavaScript Apps - Brian Noyes	W14 Exploring Microservices in a Microsoft Landscape - Marcel de Vries	W15 Learning to Live Without Data Grids in Windows 10 - Billy Hollis	W16 Conquer the Network—Making Your C# Mobile App More Resilient to Network Hiccups - Roy Cornelissen
4:30 PM	5:45 PM	W17 Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - Benjamin Day	W18 Patterns and Practices for Real-World Event-Driven Microservices - Rachel Reese	W19 Take Your Site From Ugh to OOH with Bootstrap - Philip Japikse	W20 Strike Up a Conversation with Cortana on Windows 10 - Walt Ritscher
7:00 PM	9:00 PM	VSLive's Boston By Land and Sea Tour			

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

Speakers and sessions subject to change

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

CONNECT WITH VISUAL STUDIO LIVE!

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

VSLIVE.COM/BOSTON

Develop an Azure-Connected IoT Solution in Visual Studio with C++

Marc Goodner

The Internet of Things (IoT) is everywhere nowadays. The IoT is many things, but it can most simply be understood as sensor data that's collected to the cloud for further processing. This manifests itself in many different types of devices deployed to collect that sensor data; rich cloud services for ingesting that data at scale; Big Data analytics and machine learning for processing and acting on that data; and rich reporting and client-side apps for gaining insights from this data.

So how do you get started with IoT? You need a device, a cloud and the tools to write and deploy your software solution to each.

In this article I'm going to walk you through connecting a device to Azure IoT using C++ project templates in Visual Studio, which cover the most common devices. You'll learn how to stand up an Azure IoT Hub to ingest your device data, and learn about the related services that can act on that data. Azure IoT services offer a huge

degree of flexibility, both because they're agnostic to the programming language you use on the device itself and because, through the Azure IoT device Client SDKs, they support an ever-growing number of languages and devices. In this article I'm going to focus on C++ because of the broad range of device support it offers—from the smallest, most power-efficient microcontrollers to devices that run full OSes such as Linux or Windows IoT—and because of the control it affords developers over lower-level device functionality. Even if you're not a C++ developer you'll learn enough to be able to deploy a simple application to a network-connected device and get data to Azure.

The scenario for this article is a simple one—collecting light-sensor data from a device into Azure. The goal isn't to complete an end-to-end solution, but to focus on the fundamentals—how to collect data and verify it's getting to the cloud. I'll show how this data can be used within other Azure services and how you can deploy those services within your Azure subscription. For more information on the IoT, visit aka.ms/iotmsdn16, where you'll also find links to tools discussed in this article, additional documentation, many more samples and a complete end-to-end IoT application called Connect the Dots, which you can easily deploy into your Azure subscription and to which you can connect your devices.

First, I'll set up the Azure IoT Hub, then write some code in Visual Studio to send test data to the Azure back end. Then I'll do the same with real data from a variety of devices, and conclude with a discussion of Azure services you can use with the sensor data you collect.

This article discusses:

- Setting up an Azure IoT hub
- A light sensor and three device options: Raspberry Pi, Arduino and mbed
- Using sensor data in Azure services

Technologies discussed:

Visual Studio 2015, Azure IoT Suite, Azure IoT Hub, C++, Raspberry Pi, Arduino, mbed

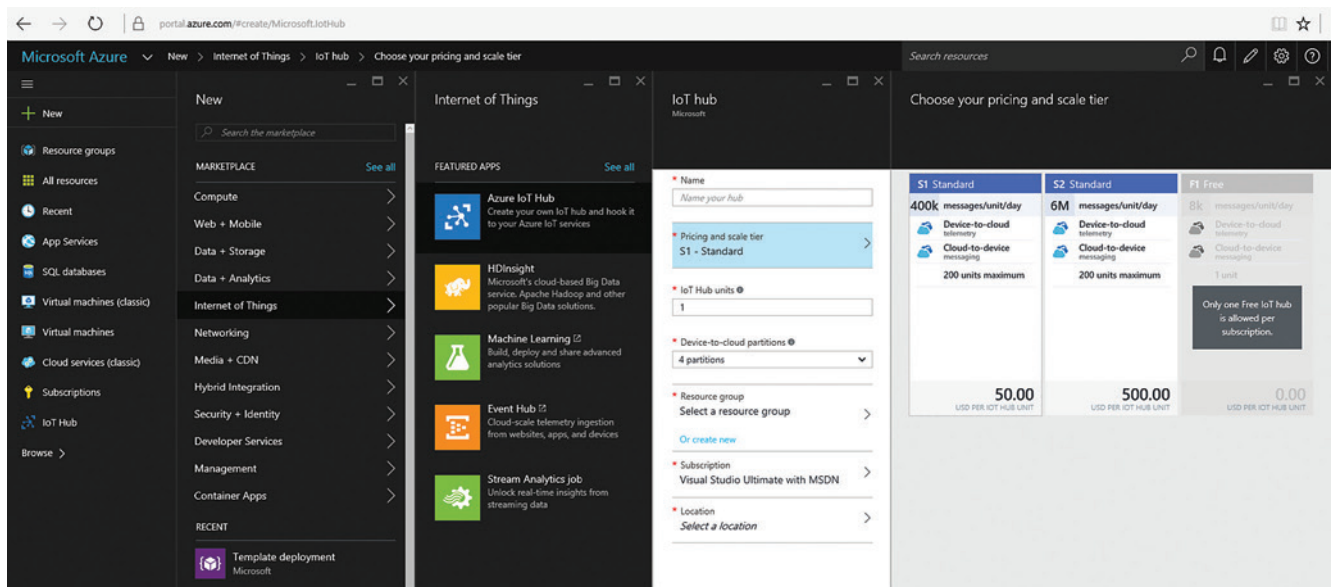


Figure 1 Creating a New Azure IoT Hub

Azure IoT Suite and Azure IoT Hub

For the cloud part of the IoT solution, Azure offers several services, including Azure IoT Hub and a bundle called Azure IoT Suite that reduces the up-front work and provides scalability in the longer term. Azure IoT Hub is the connection point for devices and is optimized to support the ingestion of large amounts of data, an essential capability given that deployments of millions of devices sending continual updates are not uncommon. You can choose a free edition that supports up to 8,000 messages per day (ideal for prototyping or personal projects), or either of two paid editions, one for smaller projects that accepts up to 400,000 messages a day and one scaled for larger implementations that supports up to 6 million messages a day. Depending on the scale of the IoT solution you're building, you can deploy as many IoT hubs as you need to support the message volume your solution requires. IoT Hub also provides capabilities for device management and provisioning tailored to the needs of IoT deployments, as well as a range of open source SDKs for developing the device connectivity and the service management parts of your solution.

The IoT is many things, but it can most simply be understood as sensor data that's collected to the cloud for further processing.

Azure IoT Suite offers a growing collection of full end-to-end IoT solutions, preconfigured to simplify the development process. These include solutions for common IoT projects such as predictive maintenance, using sensor data from machines to anticipate problems before they occur. Azure IoT Suite solutions comprise a set of preconfigured Azure IoT services, including Azure IoT

Hub, that can be provisioned into your Azure subscription so you can quickly deploy an IoT solution and then customize it to fit the unique needs of your project.

Whether you go with an Azure IoT Suite preconfigured solution or deploy your own set of Azure services, you'll be using the same Azure IoT SDKs to develop the device side of your application. In the sample that follows, I'll manually set up some simple services behind an Azure IoT Hub instance to illustrate a simple IoT end-to-end solution.

Connecting to Azure IoT Hub

The first step is to set up your Azure IoT Hub. If you don't have an Azure subscription, you can create a free trial subscription. When that's done, log into the Azure Portal and select New | Internet of Things | Azure IoT Hub, complete the options and select New. Select your pricing tier (you can deploy only one instance of the free tier of IoT Hub per subscription), complete the additional prompts and select Create (see Figure 1).

Next, make sure you have Visual Studio 2015 Community Edition or higher with the C++ tools installed and properly configured. You'll need to have the extensions for Visual C++ for Linux Development, Visual C++ for IoT Development and Azure IoT Hub Connected Service installed, as well as the Windows IoT Core templates if you plan to target a Windows IoT device. You can find these extensions on the Visual Studio Gallery or install them from within Visual Studio from the Tools | Extensions and Updates menu. Likewise, the examples for Arduino and mbed use PlatformIO to enable the compilers for those boards, which you can get at platformio.org.

Now, create a project to connect to your IoT Hub. There are many samples in the Azure IoT SDK that cover this code, each focused on a specific protocol or device. The easiest way to get this code is to install the Azure IoT Hub Connected Service extension, which currently supports C# and C++ projects. I want to quickly prototype my code before involving a device so I'm going to do this on Windows. After installing the extension, create a new Win32 project under Visual C++.

In your project, right-click on References and choose Add Connected Service. In the dialog that comes up, choose Azure IoT Hub Connected Service and select Configure (see Figure 2).

Sign in with your Azure account and you'll see your IoT Hub. Select it and choose Add. This brings up a list of existing devices if you've already added any devices to your IoT Hub; if not, it will be blank. Choose New Device and pick a name—let's call it ThingOne. Wait for it to appear and select it. This pulls down a NuGet package with the libraries you need and generates a new file, `azure_iot_hub.cpp` and corresponding header `azure_iot_hub.h`, which contains helper functions that encapsulate best practices for sending data to the Azure IoT Hub with the necessary connection string for the device you chose. Of course, you'll need to modify this generated code to handle the messages you want to send and receive. For now, we'll send the default message. Add the new header reference to your console application code and place a call to the `send_device_to_cloud_message` function in main. Your code should look like this:

```
#include "stdafx.h"
#include "azure_iot_hub.h"

int main()
{
    send_device_to_cloud_message();
    return 0;
}
```

Next, use the IoT Hub Explorer to verify your messages are going to your IoT Hub. This is a cross-platform CLI tool based on Node.js, so if it isn't installed on your system you should install it now. Get the tool via npm:

```
npm install -g iotHub-explorer@latest
```

Next, in your command prompt, start the IoT Hub Explorer to see data sent from your device into Azure IoT Hub. The necessary command with connection string and device name is in the comments of the `azure_iot_hub.cpp` file. To find this string on your own in the Azure Portal, select your IoT Hub, then choose the key icon on the Essentials bar and look in the Shared Access Policies panel that appears.

```
iotHub-explorer [<connection-string>] monitor-events ThingOne
```

You'll see a message that says events from your device are being monitored. Now switch back to Visual Studio and start debugging by hitting F5. You should see an event received in the console, "Hello Cloud."

You may want to pause here to examine the generated code handling the connection to Azure. Within the `send_device_to_cloud_message` you'll find an IoT client being created:

```
IOTHUB_CLIENT_HANDLE iotHub_client_handle =
    IoTHubClient_CreateFromConnectionString(connection_string, AMQP_Protocol);
```

Note the `AMQP_Protocol`, which is the default, though other protocols like HTTP and MQTT can also be used here. You'll

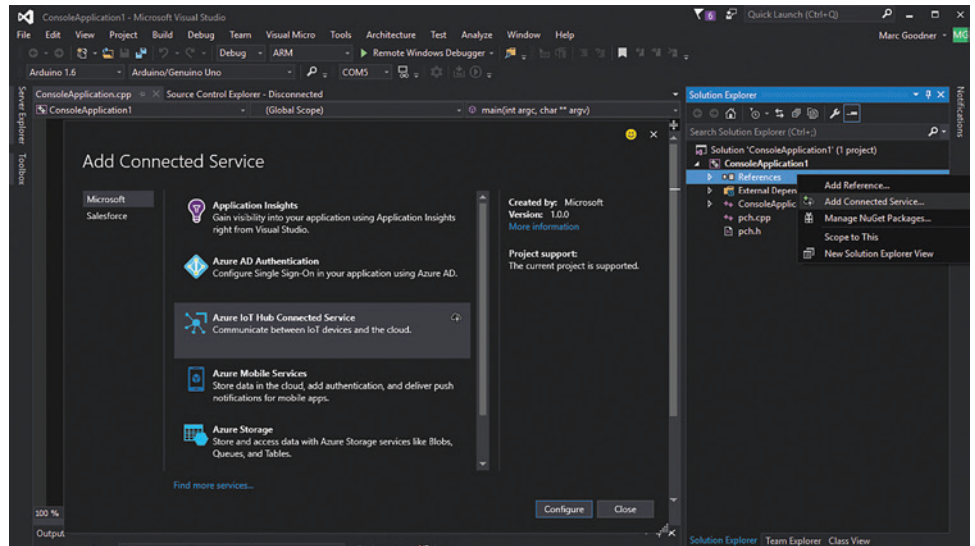


Figure 2 Using the Azure IoT Hub Connected Service Extension

also observe that error handling is provided around creating this client, and it is properly disposed. The message itself is sent just after successful creation of the client:

```
std::string message = "Hello, Cloud!";
IOTHUB_MESSAGE_HANDLE message_handle =
    IoTHubMessage_CreateFromByteArray((const unsigned char*)message.data(),
    message.size());
```

Again, you'll see that error handling is provided, and a request is sent to the IoT Hub to confirm delivery of receipt. You can now make some minor modifications to send a message of your own.

When you move on to using actual sensors, you're going to detect brightness using a photoresistor, also known as a Light Dependent Resistor or LDR, so let's modify the code to send a message with a value appropriate for that scenario. In the header file `azure_iot_hub.h`, add an `int` as a parameter of the function call. When you're done, the contents of the header should be as follows:

```
// Insert call to this function in your code:
void send_device_to_cloud_message(int);
```

In the function definition of `send_device_to_cloud_message`, change the value of `message` to use the sensor data rather than sending Hello World. Here's an example of how you can do that in JSON format:

```
char msgText[1024];
sprintf_s(msgText, sizeof(msgText), "{\"ldrValue\":%d}", ldrValue);
std::string message = msgText;
```

For more complex JSON data, or other formats, you can use your favorite C++ library.

Now update your main function to pass some values as sensor data:

```
#include "stdafx.h"
#include "azure_iot_hub.h"

int main()
{
    int ldrValue = 500; // value will be read from a sensor in later examples
    send_device_to_cloud_message(ldrValue);
    return 0;
}
```

With IoT Hub Explorer still running, watch the messages build and run your project. You'll see a new event received, with JSON-formatted data, `{ ldrValue: 500 }`.

Congratulations! You now have an IoT cloud and some code ready to wire up to a device with sensors connected to it.

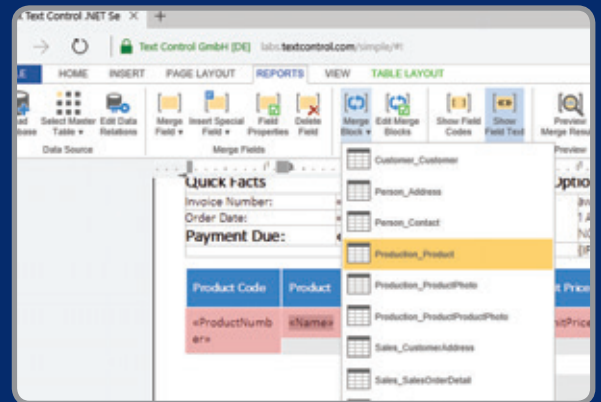
ALL ABOUT DOCUMENTS

Text Control provides deep functionality components and consulting to integrate reporting and word processing into .NET Windows, web and mobile applications.

Reporting and Mail Merge

The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor for ASP.NET, WPF and Windows Forms. Users can create documents and templates using ordinary Microsoft Word skills.

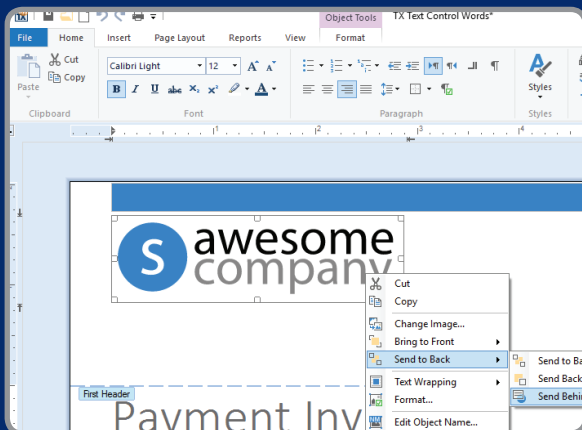
TX Text Control is completely independent from MS Word or any other third-party application and can be completely integrated into your business application.



MS Word compatible Rich Text Editing

These feature-complete, fully programmable rich edit controls offer developers a broad range of word processing features in reusable UI and non-UI components for ASP.NET, WPF and Windows Forms.

They provide comprehensive text formatting, powerful mail merge features and all word processing key concepts such as table support, images, headers and footers, page sections and spell checking.



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



Software • Training • Consulting

TEXT CONTROL

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

Devices and Sensors

For the actual sensors in this application, I'm going to use the Grove system from Seeed Studios. Grove offers a wide range of sensors with a common connector that interfaces with expansion boards connected to your device, avoiding the need for breadboards and soldering as you develop your prototype IoT solution. You can, of course, use any sensor you like, or none at all if you want to simulate sensor data within your device code. The specific Grove device I'm showing in the code samples here is a light sensor, in part for simplicity because it produces an analog output, so I don't need to find and install a specific library, a process that can vary platform to platform. In all of the examples, the sensor is plugged into the connector labeled A0 on the Grove expansion board. If you don't want to use a shield and are good with a breadboard, simply connect an LDR to the power and an analog input pin on your board.

For the device part of the solution, I'm going to cover three alternatives: Raspberry Pi, Arduino and mbed. There are many factors to take into account when choosing a device for your IoT project, and these three options cover a range of different scenarios. For smaller deployments, a variety of off-the-shelf devices can simplify the development process. However, for large deployments, these choices aren't as cost-effective. (For a deployment of tens of thousands of devices, reducing hardware costs, even by as little as a dollar or two, can make a big difference.) If you're just getting started learning about the IoT, simplicity of development probably trumps minimizing such costs, but for prototyping you'll probably want to consider it. Basic setup of the devices isn't covered here, but you'll find pointers to materials online at aka.ms/iotmsdnb16. **Figure 3** shows the devices and sensor discussed in this section.

At the time of this writing, the Azure IoT Hub Connected Service extension can only pull down the binary libraries for use on Windows. Microsoft is working to enable this for targeting other platforms, as well. That said, the code the extension generates does work across platforms. In the steps that follow, you'll reuse the code generated earlier for handling messages to the Azure IoT Hub. Each device section is intended to stand on its own, so you can skip sections for devices you're not interested in.

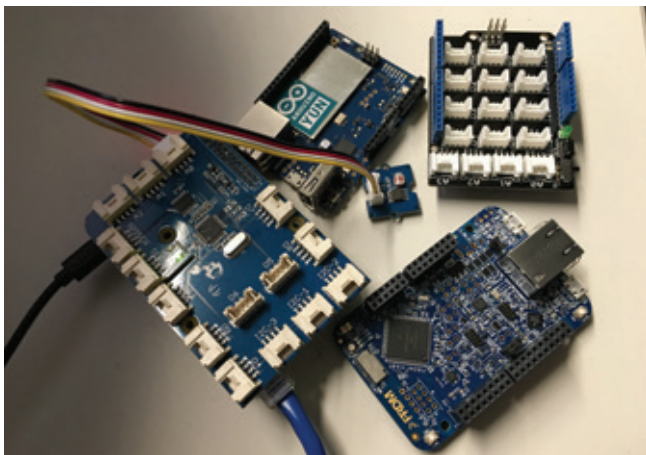


Figure 3 From the Left, a Raspberry Pi 2 with a Grove Hat and Light Sensor Attached, an Arduino Yun, a Grove Base Shield, and the FRDM 64 mbed Board at Bottom

Figure 4 Reading Sensor Data Using a Raspberry Pi

```
#include "azure_iot_hub.h"
#include <unistd.h>
#include "grovepi.h"

int main()
{
    int ldrValue;

    while(1) {
        ldrValue = analogRead(0);
        send_device_to_cloud_message(ldrValue);
        sleep(5);
    }
    return 0;
}
```

Raspberry Pi The Raspberry Pi is a single-board computer with an ARM processor and a GPU. It can run various flavors of Linux and the Raspberry Pi 2 also supports Windows 10 IoT. Raspberry Pi offers lots of flexibility—it's a full computer and includes networking, USB, GPIO and HDMI (though it doesn't support analog sensors), and thanks to a vibrant maker community, example projects abound and help is available online. That flexibility does come at a cost, though—a higher power draw compared to alternatives and limited options to customize. So if you're prototyping a device that won't have access to hardwired power or will be deployed in large numbers, you might want to start with another board.

For this article, I'm going to use a Raspberry Pi in conjunction with an expansion board for Grove connectors from Dexter Industries, which also adds analog support, and a light sensor connected to A0. I'm also going to use the recently launched new project system support for Linux (part of the Visual C++ for Linux Development extension), which lets you build and debug code on remote Linux machines. Go to File | New Project. Under Visual C++ | Cross Platform, select Console Application (Linux). This opens a help file that explains the new capabilities of this project type and tells how to add your remote Linux machine to the project. The instructions also cover acquisition of the Azure IoT libraries for Linux and some tips for setting up the Grove shield.

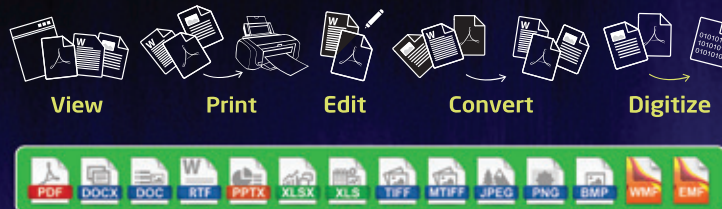
Copy the generated files from the first project into this project. Open the cpp file in the solution named after your project and add the reference to `azure_iot_hub.h`. Now update the code in `main` to read the sensor data from the device continuously, as shown in **Figure 4**.

Now you can run your application from within Visual Studio. It will copy your sources to the Pi in the location you specified, build and run them. As with any other code in Visual Studio, you can set breakpoints. Once the code runs, you'll see an event received by the IoT Hub Explorer as with the simulated data, `{ldrValue: 1013}`.

This support for Linux isn't tied to the Raspberry Pi; it also works with other Linux-based devices like the Intel Edison, the Beagle-Bone or any remote Linux machine.

Windows 10 IoT Core on the Raspberry Pi 2 is another option. See windowsondevices.com for information on getting started. After the tools are installed, go to File | New Project, then look under Visual C++ Windows and select Blank App (Universal Windows). Add a reference to Windows IoT Extensions for the UWP to the project. At the time of writing, there are some additional steps required for UWP. Please see the comments in the generated file `azure_iot_hub.cpp`

One component suite for all Document requirements



See us at the
**Build 2016
Expo**



WinForms, WPF, Web Forms, MVC, HTML5



Download your **FREE**
30-day trial today!!!

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



XtremeDocumentStudio .NET

in case those are still required. In the application code of MainPage.xaml.cpp, add the reference to the azure_iot_hub.h include file.

In the MainPage.xaml.h include file, add the following private declarations:

```
private:
    const int readPin = 4;
    Windows::Devices::Gpio::GpioPin ^pin_;
```

This code illustrates how to read directly from a GPIO pin using the UWP APIs. Please see aka.ms/iotmsdnb16 for updates regarding using the Grove Pi shield with C++. Update the function definition and implementation to pass the sensor values as a JSON object as in the earlier example.

Next, you'll update the implementation of MainPage in MainPage.xaml.cpp, as follows:

```
MainPage::MainPage()
{
    int ldrValue;
    auto gpio = GpioController::GetDefault();
    pin_ = gpio->OpenPin(readPin);
    pinValue = (int) pin_->Read();
    send_device_to_cloud_message(ldrValue);
}
```

Make sure the Azure IoT Hub is listening for your device, then build and deploy. You'll see the JSON messages with the value from your light sensor stream by. Debugging works just as you'd expect because the target device is another Windows machine. There's no special configuration required when your target is a Windows IoT device.

Arduino The Arduino is found in more maker projects than probably any other board out there. As a result of its widespread appeal, many other providers have added support for their boards to the Arduino IDE. Among sensor manufacturers, the level of support is similarly high, with libraries that can easily be incorporated into the Arduino IDE. The boards themselves have common connectors that provide a variety of digital and analog inputs and outputs in a common form factor for expansion shields to be added, such as the Grove base shield. That form factor has been adopted by many other board providers. There are many Arduino/Genuino boards to choose from, ranging from low-power MCUs to powerful ARM chips, and a rich ecosystem of third-party boards is available. As an open source hardware platform, all the schematics are available, making this an excellent platform for use in larger deployments.

Not all Arduinos have networking built in, though, so you'll want to either choose one that does or use an expansion shield that adds networking to your device. Attach a Grove base shield and plug the light sensor into the port labeled A0.

Figure 5 Reading Sensor Data Using Mbed

```
#include "azure_iot_hub.h"
#include "mbed.h"

AnalogIn lightSensorPin(A0);
float ldrSensorValue;
int ldrValue

int main()
{
    while(1) {
        ldrSensorValue = lightSensorPin.read();
        ldrValue = (int) (ldrSensorValue * 1000);
        send_device_to_cloud_message(ldrValue);
        delay(2000);
    }
}
```

Basic Arduino projects are called sketches and comprise a folder containing a file of the same name ending in .ino. This file is just a C++ file. The Arduino IDE hides a lot of complexity of its default Wiring API libraries. There are many guides online that cover basic Arduino setup and usage; visit [Arduino.cc](https://www.arduino.cc) or aka.ms/iotmsdnb16 if you need pointers to those. If you want the fuller functionality of Visual Studio but want to stay within the Arduino ecosystem and use the .ino format, Visual Micro provides an extension that brings most of the board and library management functions from the Arduino IDE directly into Visual Studio.

To use C++ directly rather than the .ino sketch format, make sure you've installed the Visual C++ for IoT Development extension and create a new project. Then, from Visual C++ | Cross Platform, select Blank Application (IoT) project. This opens a help file that explains the new capabilities of this project type. Refer to those instructions for how to add and configure your board within the project.

Refer to the readme in the project on initializing PlatformIO for use with the Arduino and importing the required Azure IoT libraries. Copy the azure_iot_hub.h file from the first exercise into the lib folder and the azure_iot_hub.cpp into the src folder:

```
#include "azure_iot_hub.h"

int lightSensorPin = 0;
int ldrValue;

void loop()
{
    ldrValue = analogRead(lightSensorPin);
    send_device_to_cloud_message(ldrValue);
    delay(2000);
}
```

Now you can build and deploy your code to the board. Make sure the Azure IoT Hub Explorer is running so you can see the messages being sent from the board. Debugging is supported, as well. Set up for this varies, so please see the help file that's included in the project.

mbed The ARM mbed platform is another excellent choice. This is an OS for ARM-based microcontrollers that provides a common framework for development and is supported by a wide range of prototyping boards supplied by the many ARM silicon vendors. These vendors all have their own path to production of custom designs. For deployments at scale, mbed is an excellent place to start. At Microsoft, we've been working with boards from Freescale, Marvel, STMicro and others. The range of choice here, however, can also make it a little hard to know where to start, but the excellent community online at mbed.org can help. Be aware that you won't find as many maker-style projects online for mbed as for Raspberry Pi or Arduino. Every ARM vendor who produces an mbed board has getting started guidance; I highly recommend you start there.

For this article, I'm going to use the Freescale Freedom K64F board because it has built-in Ethernet, which makes it a snap to get connected to the Internet. This board also has headers matching the Arduino so you can use the same Grove base shield for connecting your sensors. Attach the light sensor to the plug labeled A0.

Again, as with the Raspberry Pi and Arduino, I'm going to use the microcontroller support in Visual Studio. Make sure you've installed the Visual C++ for IoT Development extension and create a new project. Select Visual C++ | Cross Platform | Blank Application (IoT). This opens a help file that explains the new capabilities



Great Apps Happen By Design



Design Desktop Web Mobile Data Visualization

Superior user experiences start with Infragistics Ultimate.

Choose UX & UI tools built to accelerate the application design and development process.

Download a free trial of Infragistics Ultimate at
infragistics.com

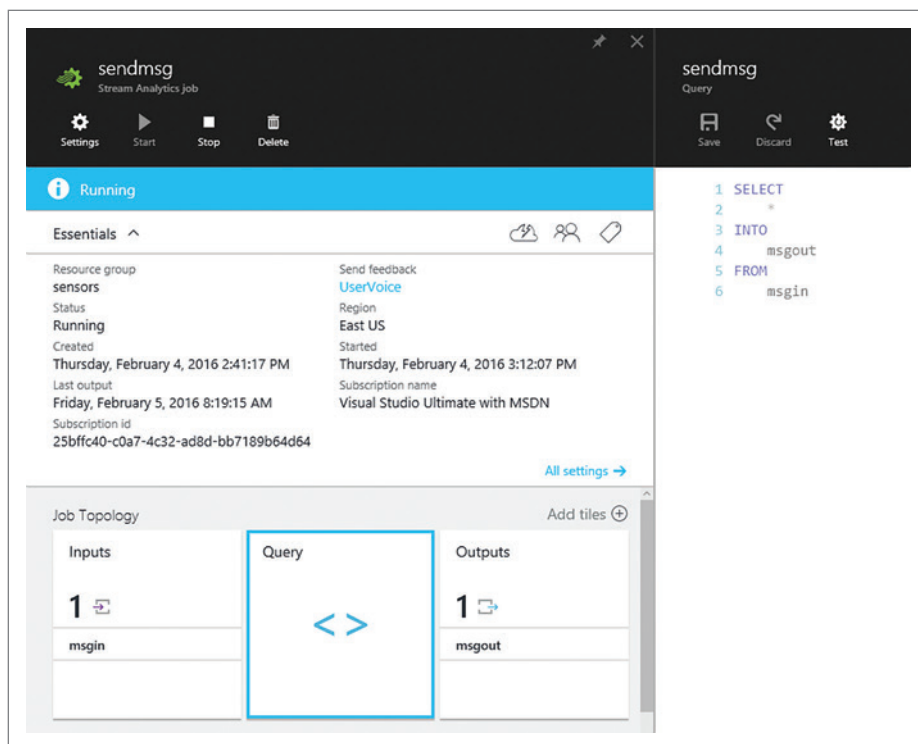


Figure 6 The Simplest Stream Analytics Job Ever

of this project type. Refer to those instructions for how to add and configure your board within the project.

I'm going to use PlatformIO as my build tool for this project. PlatformIO is a flexible system that supports many different microcontroller platforms. Please refer to the readme in the project on initializing PlatformIO for use with an mbed target and importing the required Azure IoT libraries. Copy the `azure_iot_hub.h` file from the first exercise into the `lib` folder and the `azure_iot_hub.cpp` into the `src` folder.

Update your program code as shown in **Figure 5**.

Now you can build and deploy your code to the board. Make sure the Azure IoT Hub Explorer is running so you can see the messages being sent from the board. Debugging is supported, as well. Setting this up varies so please see the help file that's included in the project.

Using Your Sensor Data in Azure Services

Now that you've gotten the sensor data flowing from your device to Azure, you can use it with any available Azure service. This is where the reference implementations in the Azure IoT Suite or the Connect the Dots project can help you see the possibilities and how things are connected together.

As an example, you can process real-time information coming in via IoT Hubs and use it as input to other services. A common pattern here is anomaly detection and notification. To configure this type of alert, you'd create a Stream Analytics service that takes as its input your IoT Hub. Next, you'd define an output for this job, for example to Storage Blob or PowerBI, and then define a query that parses data coming from the input, looking for values above a certain threshold, for example. Any matches would then be sent to the defined output (see **Figure 6**).

You could use the Storage Blob as an input to a logic app that would then send an SMS alert, or in the case of PowerBI it could be used in a near-real-time dashboard of changing conditions. Machine learning services also provide rich opportunities to unlock value from the vast amounts of sensor data that can be collected, either in real-time hot paths or by analyzing historical data. These are just a few examples. With all of the Azure services available to further process and report on your data, you can realize any of your IoT needs.

One thing to note is that IoT Hubs can be used anywhere you can use Event Hubs. If you find a service you're trying to use doesn't have an IoT Hub option, you'll need to get the Event Hub compatible name. Go to your IoT Hub in the Azure Portal and select Messaging. Under Device to Cloud settings, you'll see the Event Hub compatible name and endpoint.

Taking full advantage of IoT data always involves the use of multiple services. Azure Resource Manager (ARM) templates can help you manage that complexity. These templates, written in JSON, define Azure services for deployment, including the ordering of deployment, location, values to use, and so forth, and they allow you to quickly deploy a full set of services to create an IoT cloud back end. These templates are used within the IoT Suite for deploying solutions into specific Azure subscriptions. Because the templates are simple JSON files, you can also keep track of them in your version control system for use in future deployments, say moving from prototype to production deployment.

Your Future with IoT

You have a role to play in building the Internet of Things. If you haven't already started down this path, hopefully this article has piqued your interest. If you've been hacking Arduinos and Raspberry Pi devices for years already, perhaps you'll find value in the new Visual Studio C++ capabilities introduced here. If you aren't a C++ developer, you may find that these new capabilities make C++ far more approachable.

There is much more to Azure IoT than I've presented here. I covered the basics to get you started, and provided pointers for you to go further. Do give it a try. I look forward to seeing what you build.

Remember to visit aka.ms/iotmsdnb16 for additional resources and to keep learning. ■

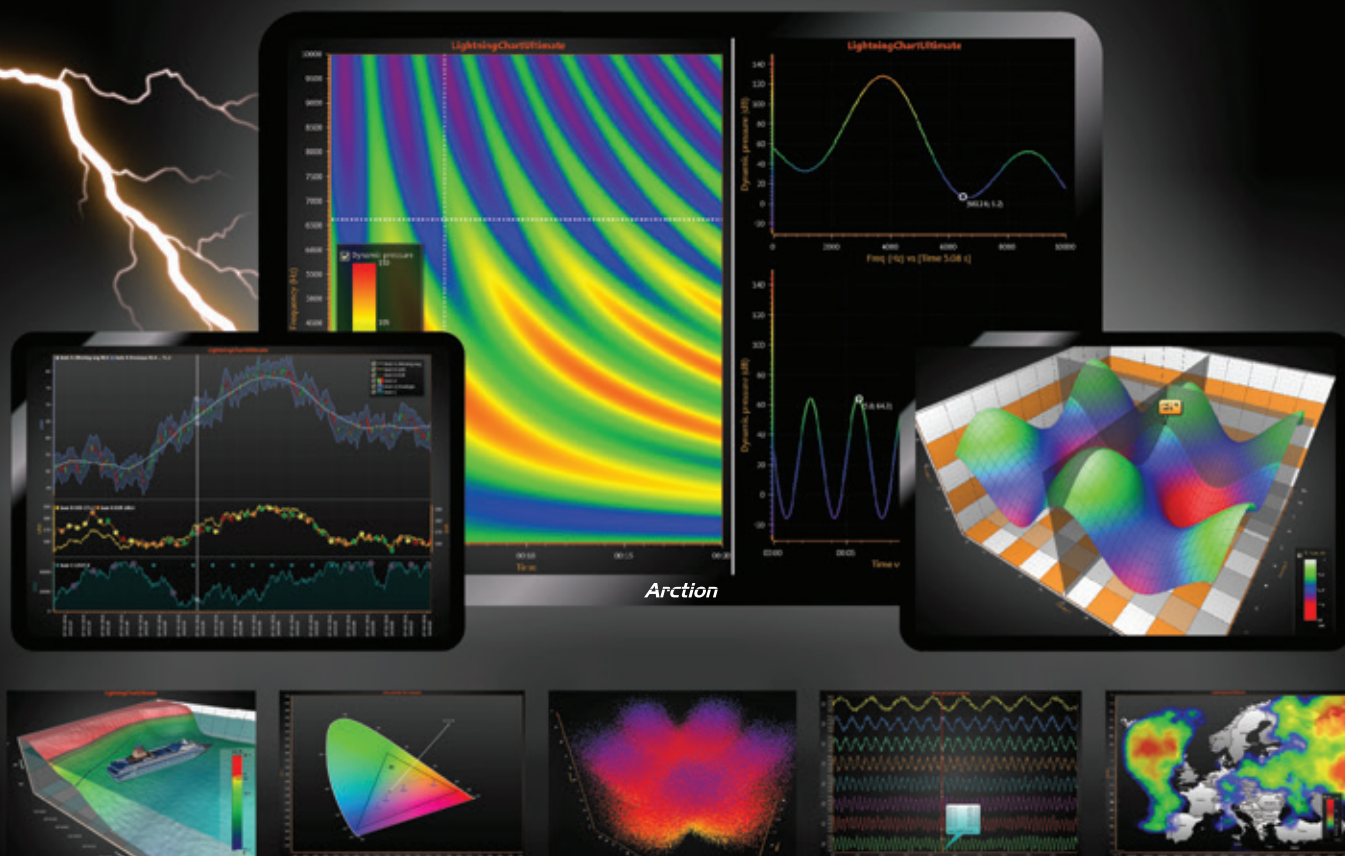
MARC GOODNER is a program manager focused on IoT tooling on the Visual Studio team and an active participant in Microsoft's Garage Maker Community. He appreciates a fine single malt.

THANKS to the following Microsoft technical experts for reviewing this article: Olivier Bloch and Artur Laksberg

**Ultimate Data Visualization
Controls for WPF and WinForms...**

LightningChart v.7

New! DirectX 11 rendering - Faster WPF Chart - Bindable WPF Chart - Smith Chart



LIGHTNING-FAST CHARTING COMPONENTS FOR SCIENCE, ENGINEERING AND TRADING

- Superior rendering performance
- Outstanding configurability
- DirectX 9 and 11 rendering engines
- WARP rendering for virtual machines
- Optimized for real-time data monitoring
- Touch-enabled operations
- Supports gigantic data sets
- On-line and off-line maps
- Great customer support
- Hundreds of examples

4-in-1: All editions included

Prefer performance or binding features

WinForms

WPF

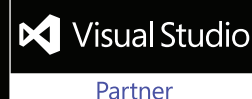
WPF properties binding

WPF properties + data binding

Performance



Download a free 30-day trial
www.LightningChart.com



VOTE "YES" FOR BETTER

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

CODE Washington, D.C.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS



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

SUPPORTED BY



Visual Studio
MAGAZINE



PRODUCED BY



WASHINGTON, D.C. • OCT. 3-6, 2016

RENAISSANCE, WASHINGTON, D.C.



CONNECT WITH VISUAL STUDIO LIVE!



twitter.com/vslive – @VSLive



facebook.com – Search “VSLive”



linkedin.com – Join the “Visual Studio Live” group!

DEVELOPMENT TOPICS INCLUDE:

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

Do your developer duty: register to join us today

REGISTER NOW AND SAVE \$300!



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

USE PROMO CODE VSLDC2

VSLIVE.COM/DC



Logging with .NET Core

In the February issue, I delved into the new configuration API included in the newly named .NET Core 1.0 platform (see bit.ly/10oqmkJ). (I assume most readers have heard about the recently renamed .NET Core 1.0, which was formerly referred to as .NET Core 5 and part of the ASP.NET 5 platform [see bit.ly/10oq7Wl].) In that article I used unit testing in order to explore the `Microsoft.Extensions.Configuration` API. In this article I take a similar approach, except with `Microsoft.Extensions.Logging`. The key difference in my approach is that I'm testing it from a .NET 4.6 CSProj file rather than an ASP.NET Core project. This emphasizes the fact that .NET Core is available for you to consider using immediately—even if you haven't migrated to ASP.NET Core projects.

What Microsoft is providing with `Microsoft.Extensions.Logging` is that wrapper so everyone doesn't have to write their own.

Logging? Why on earth do we need a new logging framework? We already have NLog, Log4Net, Loggr, Serilog and the built-in `Microsoft.Diagnostics.Trace/Debug/TraceSource`, just to name a few. As it turns out, the fact that there are so many logging frameworks is actually one of the driving factors that make `Microsoft.Extensions.Logging` relevant. As a developer faced with the myriad of choices, you're likely to select one knowing you might have to switch to another one later. Therefore, you're probably tempted to write your own logging API wrapper that invokes whichever particular logging framework you or your company chooses this week. Similarly, you might use one particular logging framework in your application, only to find that one of the libraries you're leveraging is using another, causing you to have to write a listener that takes the messages from one to the other.

What Microsoft is providing with `Microsoft.Extensions.Logging` is that wrapper so everyone doesn't have to write their own. This wrapper provides one set of APIs that are then forwarded to a provider of your choosing. And, while Microsoft includes providers for

things like the Console (`Microsoft.Extensions.Logging.Console`), debugging (`Microsoft.Extensions.Logging.Debug`), the event log (`Microsoft.Extensions.Logging.EventLog`) and `TraceSource` (`Microsoft.Extensions.Logging.TraceSource`), it has also collaborated with the various logging framework teams (including third parties like NLog, Serilog, Loggr, Log4Net and more) so that there are `Microsoft.Extensions.Logging` compatible providers from them, too.

Getting Started

The root of the logging activity begins with a log factory, as shown in **Figure 1**.

As the code demonstrates, to begin you instantiate a `Microsoft.Extensions.Logging.LoggerFactory`, which implements `ILoggerFactory` in the same namespace. Next, you specify which providers you want to utilize by leveraging the extension method of `ILoggerFactory`. In **Figure 1**, I specifically use `Microsoft.Extensions.Logging.ConsoleLoggerExtensions.AddConsole` and `Microsoft.Extensions.Logging.DebugLoggerFactoryExtensions.AddDebug`. (Although the classes are both in the `Microsoft.Extensions.Logging` namespace, they're actually found in the `Microsoft.Extensions.Logging.Console` and `Microsoft.Extensions.Logging.Debug` NuGet packages, respectively.)

The extension methods are simply convenient shortcuts for the more general way to add a provider—`ILoggerFactory.AddProvider(ILoggerProvider provider)`. The shortcut is that the `AddProvider` method requires an instance of the log provider—likely one whose constructor requires a log-level filter expression—while the extension methods provide defaults for such expressions. For example, the constructor signature for `ConsoleLoggerProvider` is:

```
public ConsoleLoggerProvider(Func<string, LogLevel, bool> filter,
    bool includeScopes);
```

This first parameter is a predicate expression that allows you to define whether a message will appear in the output based on the value of the text logged and the log level.

Figure 1 How to Use `Microsoft.Extensions.Logging`

```
public static void Main(string[] args = null)
{
    ILoggerFactory loggerFactory = new LoggerFactory()
        .AddConsole()
        .AddDebug();

    ILogger logger = loggerFactory.CreateLogger<Program>();

    logger.LogInformation(
        "This is a test of the emergency broadcast system.");
}
```

Code download available at [GitHub.com/IntelliTect/Articles](https://github.com/IntelliTect/Articles).

Whitepaper Available
Four Ways to Optimize ASP.NET Performance

Extreme Performance Linear Scalability

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

In-Memory Distributed Cache

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

ASP.NET Optimization in Web Farms

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

Full Integration with Microsoft Visual Studio

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



sales@alachisoft.com

US: +1 (925) 236 3830

FREE Download
www.alachisoft.com

For example, you could call `AddProvider` with a specific `ConsoleLoggerProvider` instance that was constructed from a filter of all messages higher (more significant) than `LogLevel.Information`:

```
loggerFactory.AddProvider(
    new ConsoleLoggerProvider(
        (text, logLevel) => logLevel >= LogLevel.Verbose, true));
```

(Interestingly, unlike the extension methods that return an `ILoggerFactory`, `AddProvider` returns `void`—preventing the fluid type syntax shown in **Figure 1**.)

It's important to be cognizant that, unfortunately, there's some inconsistency between log providers as to whether a high log-level value is more or less significant. Does a log level of 6 indicate a critical error occurred or is it just a verbose diagnostic message? `Microsoft.Extensions.Logging.LogLevel` uses high values to indicate higher priority with the following `LogLevel` enum declaration:

```
public enum LogLevel
{
    Debug = 1,
    Verbose = 2,
    Information = 3,
    Warning = 4,
    Error = 5,
    Critical = 6,
    None = int.MaxValue
}
```

Therefore, by instantiating a `ConsoleLoggerProvider` that writes messages only when the `logLevel >= LogLevel.Verbose`, you're excluding only `Debug`-level messages from being written to the output.

Because the value `Program` indicates class-level context, you'll likely want to instantiate a different logger instance for each class from which you want to log.

Note that you can add multiple providers to the log factory, even multiple providers of the same type. Therefore, if I add an invocation of `ILoggerFactory.AddProvider` to **Figure 1**, a call to `ILogger.LogInformation` would display a message on the console twice. The first console provider (the one added by `AddConsole`) defaults to displaying anything `LogLevel.Information` or higher. However, an `ILogger.LogVerbose` call would appear only once as only the second provider (added via the `AddProvider` method) would successfully avoid being filtered out.

Logging Patterns

As **Figure 1** demonstrates, the root of all logging begins with a log factory from which you can request an `ILogger` via the `ILoggerFactory.CreateLogger<T>` method. The generic type `T` in this method is to identify the class in which the code executes, so it's possible to write out the class name in which the logger is writing messages. In other words, by calling `loggerFactory.CreateLogger<Program>`, you essentially initiate a logger specific to the `Program` class so that each time a message is written, it's also possible to write the

execution context as being within the `Program` class. Thus, the console output of **Figure 1** is:

```
info: SampleWebConsoleApp.Program[0]
      This is a test of the emergency broadcast system.
```

This output is based on the following:

- “info” results from the fact that this is a `LogInformation` method call.
- “SampleWebConsoleApp.Program” is determined from `T`.
- “[0]” is the `eventId`—a value I didn't specify so it defaults to 0.
- “This is a test of the emergency broadcast system.” is the messages argument passed to `LogInformation`.

Because the value `Program` indicates class-level context, you'll likely want to instantiate a different logger instance for each class from which you want to log. For example, if `Program` creates and calls into a `Controller` class instance, you'll want to have a new logger instance within the `Controller` class that was created via another method call where `T` is now `Controller`:

```
loggerFactory.CreateLogger<Controller>()
```

As you may notice, this requires access to the same logger factory instance on which the providers were previously configured. And while it's conceivable you could pass the logger factory instance into every class from which you want to perform logging, it would quickly become a hassle that would beg for refactoring.

The solution is to save a single static `ILoggerFactory` as a static property that's available for all classes when instantiating their object's specific `ILogger` instance. For example, consider adding an `ApplicationLogging` static class that includes a static `ILoggerFactory` instance:

```
public static class ApplicationLogging
{
    public static ILoggerFactory LoggerFactory { get; } = new LoggerFactory();
    public static ILogger CreateLogger<T>() =>
        LoggerFactory.CreateLogger<T>();
}
```

The obvious concern in such a class is whether the `LoggerFactory` is thread-safe. And, fortunately, as the `AddProvider` method shown in **Figure 2** demonstrates, it is.

Because the only data in the `ILogger` instance is determined from the generic type `T`, you might argue that each class could have a static `ILogger` that each class's object could leverage. However, assuming the programming standard of ensuring thread safety for all static members, such an approach would require concurrency control within the `ILogger` implementation (which isn't there by default), and likely result in a significant bottleneck as locks are taken and released. For this reason, the recommendation, in fact, is to have an individual `ILogger` instance for each instance of a class. The result, therefore, is an `ILogger` property on each class for which you wish to support logging (see **Figure 3**).

Figure 2 The `Microsoft.Extensions.Logging.LoggerFactory` `AddProvider` Implementation

```
public void AddProvider(ILoggerProvider provider)
{
    lock (_sync)
    {
        _providers = _providers.Concat(new[] { provider }).ToArray();
        foreach (var logger in _loggers)
        {
            logger.Value.AddProvider(provider);
        }
    }
}
```




R#

—

ReSharper

THE LEGENDARY EXTENSION
TO VISUAL STUDIO*

jetbrains.com/resharper

JET
BRAINS

—

THE DRIVE
TO DEVELOP

* WARNING! PROLONGED USE
MAY CAUSE ADDICTION



Understanding Scopes

Frequently, providers support the concept of “scope” such that you could (for example) log how your code traverses a call chain. Continuing the example, if Program invokes a method on a Controller class, that class in turn instantiates its own logger instance with its own context of type T. However, rather than simply displaying a message context of info: SampleWebConsoleApp.Program[0] followed by info: SampleWebConsoleApp.Controller[0], you might wish to log that Program-invoked Controller and possibly even include the method names themselves. To achieve this,

Figure 3 Adding an ILogger Instance to Each Object That Needs Logging

```
public class Controller
{
    ILogger Logger { get; } =
        ApplicationLogging.CreateLogger<Controller>();

    // ...
    public void Initialize()
    {
        using (Logger.BeginScopeImpl(
            $"=> { nameof(Initialize) }"))
        {
            Logger.LogInformation("Initialize the data");
            //...
            Logger.LogInformation("Initialize the UI");
            //...
        }
    }
}
```

Figure 4 An Updated Implementation of Program

```
public class Program
{
    static ILogger Logger { get; } =
        ApplicationLogging.CreateLogger<Program>();

    public static void Main(string[] args = null)
    {
        ApplicationLogging.LoggerFactory.AddConsole(true);
        Logger.LogInformation(
            "This is a test of the emergency broadcast system.");
        using (Logger.BeginScopeImpl(nameof(Main)))
        {
            Logger.LogInformation("Begin using controller");
            Controller controller = new Controller();
            controller.Initialize();
            Logger.LogInformation("End using controller");
        }
        Logger.Log(LogLevel.Information, 0, "Shutting Down...", null, null);
    }
}
```

Figure 5 Console Logging Output with Scopes Included

```
info: SampleWebConsoleApp.Program[0]
      This is a test of the emergency broadcast system.
info: SampleWebConsoleApp.Program[0]
      => Main
      Begin using controller
info: SampleWebConsoleApp.Controller[0]
      => Main => Initialize
      Initialize the data
info: SampleWebConsoleApp.Controller[0]
      => Main => Initialize
      Initialize the UI
info: SampleWebConsoleApp.Program[0]
      => Main
      End using controller
info: SampleWebConsoleApp.Program[0]
      Shutting down...
```

you activate the concept of scope within the provider. Figure 3 provides an example within the Initialize method via the invocation of Logger.BeginScopeImpl.

Using the logging pattern while leveraging the scope activation will result in a Program class that might look a little like Figure 4.

The output of Figure 3 combined with Figure 4 is shown in Figure 5.

Notice how the scope automatically unwinds to no longer include Initialize or Main. This functionality is provided by the fact that BeginScopeImpl returns an IDisposable instance that automatically unwinds the scope when the using statement calls Dispose.

Leveraging a Third-Party Provider

To make available some of the most prominent third-party logging frameworks, Microsoft collaborated with its developers and ensured there are providers for each. Without indicating a preference, consider how to connect up the NLog framework, as demonstrated in Figure 6.

Of course, one of
the most common reasons
to log is to record when an
exception is thrown.

Most of this code is well-known to those familiar with NLog. First, I instantiate and configure an NLog target of type NLog.Targets.MemoryTarget. (There are numerous NLog targets and each can be identified and configured in the NLog configuration file, in addition to using configuration code as shown in Figure 6.) Notice that while similar in appearance, the Layout is assigned a literal value of \${message}, not a string interpolated value.

Once added to the LoggerFactory and configured, the code is identical to any other provider code.

Exception Handling

Of course, one of the most common reasons to log is to record when an exception is thrown—more specifically, when the exception is being handled rather than re-thrown or when the exception is entirely unhandled (see bit.ly/1LYGBVS). As you'd expect, Microsoft.Extensions.Logging has specific methods for handling an exception. Most such methods are implemented in Microsoft.Extensions.Logging.LoggerExtensions as extension methods to ILogger. And, it's from this class that each method specific to a particular log level (ILogger.LogInformation, ILogger.LogDebug, ILogger.LogCritical and so forth) is implemented. For example, if you want to log a LogLevel.Critical message regarding an exception (perhaps before gracefully shutting down the application), you'd call:

```
Logger.LogCritical(message,
    new InvalidOperationException("Yikes..."));
```

Another important aspect of logging and exception handling is that logging, especially when handling exceptions, should not throw an exception. If an exception is thrown when you log,

Figure 6 Configuring NLog as
a Microsoft.Extensions.Logging Provider

```
[TestClass]
public class NLogLoggingTests
{
    ILogger Logger {get;}
    = ApplicationLogging.CreateLogger<NLogLoggingTests>();

    [TestMethod]
    public void LogInformation_UsingMemoryTarget_LogMessageAppears()
    {
        // Add NLog provider
        ApplicationLogging.LoggerFactory.AddNLog(
            new global::NLog.LogFactory(
                global::NLog.LogManager.Configuration));

        // Configure target
        MemoryTarget target = new MemoryTarget();
        target.Layout = "${message}";
        global::NLog.Config.SimpleConfigurator.ConfigureForTargetLogging(
            target, global::NLog.LogLevel.Info);

        Logger.LogInformation(Message);
        Assert.AreEqual<string>(
            Message, target.Logs.FirstOrDefault<string>());
    }
}
```

presumably the message or exception will never get written and could potentially go entirely unnoticed, no matter how critical. Unfortunately, the out-of-the-box ILogger implementation—Microsoft.Extensions.Logging.Logger—has no such exception handling, so if an exception does occur, the calling code would need to handle it—and do so every time Logger.LogX is called. A general approach to solving this is to possibly wrap Logger so as to catch the exception. However, you might want to implement your own versions of ILogger and ILoggerFactory (see bit.ly/1LYHq0Q for an example). Given that .NET Core is open source, you could even clone the class and purposely implement the exception handling in your very own LoggerFactory and ILogger implementations.

Wrapping Up

I started out by asking, “Why on Earth would we want yet another logging framework in .NET?” I hope by now this is clear. The new framework creates an abstraction layer or wrapper that enables you to use whichever logging framework you want as a provider. This ensures you have the maximum flexibility in your work as a developer. Furthermore, even though it’s only available with .NET Core, referencing .NET Core NuGet packages like Microsoft.Extensions.Logging for a standard Visual Studio .NET 4.6 project is no problem. ■

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, Chris Finlayson and Michael Stokesbary

msdnmagazine.com



dtSearch®

Instantly Search Terabytes of Text

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

Highlights hits in all data types;
25+ search options

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

Visit dtSearch.com for

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

The Smart Choice for Text Retrieval®
since 1991

dtSearch.com 1-800-IT-FINDS



Writing UWP Apps for the Internet of Things

One of the most-used phrases in the technology industry today is “the Internet of Things,” often abbreviated as IoT. The IoT promises to turn every device into a smart device by connecting it to the cloud. From the cloud, a device can provide a control surface and raw data. Cameras can be controlled remotely. Data can be collected and analyzed for patterns and insight.

While there have been many articles in *MSDN Magazine* on how to collect and analyze data from these devices, there hasn't yet been any discussion from hardware or wiring perspectives. However, jumping in with both feet into IoT might require developers to acquire new skills such as electronics design, electricity and, in some cases, soldering. Developers, by nature, are quite comfortable writing code but might not feel quite so comfortable with the circuits and electrons underpinning everything in the virtual world. Many software developers might find themselves wondering what to do with solderless breadboards, jumper cables and resistors. This column will explain their purpose.

Of course, programmable devices have existed for many years. Writing code for these devices, however, often required extensive knowledge of proprietary toolsets and expensive prototyping hardware. The Raspberry Pi 2 Model B can run Windows 10 IoT Core, a special version of Windows 10. Windows 10 IoT Core is a free download from the Windows Dev Center IoT Web site at dev.windows.com/iot. Now that Windows 10 IoT Core runs on the Raspberry Pi 2, Universal Windows Platform (UWP) developers can leverage their existing code and skills.

In this column, I'll create a UWP app that runs on the Raspberry Pi 2 and will flash an LED light based on the data from a weather API. I'll introduce IoT concepts, the Raspberry Pi 2 Model B hardware and how to control it from C# code.

Project: Monitoring for Frost

As spring brings back warm weather, many eagerly await the chance to start gardening again. However, early spring in many areas can also bring a few cold weather snaps. Frost can seriously damage plants, so as a gardener, I want to know if cold weather is in the forecast. For this, I'll display a message on the screen if the forecast low goes below 38 degrees Fahrenheit (3.3 degrees Celsius). The app will also rapidly flash an LED as an extra warning.

Code download available at bit.ly/1PQyT12.

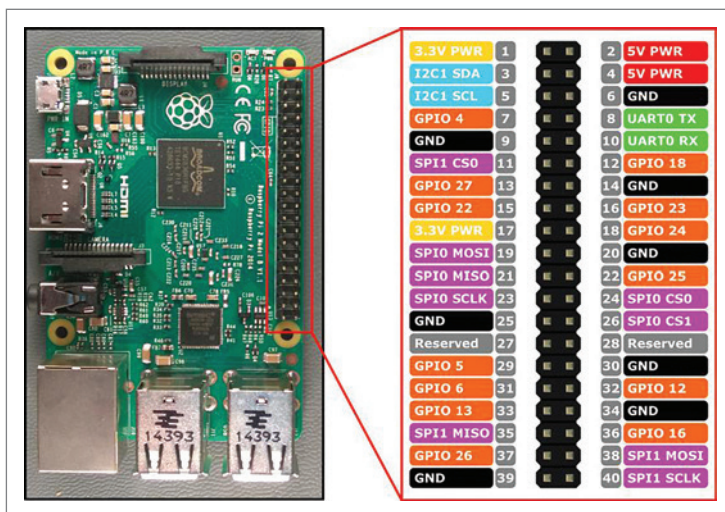


Figure 1 Raspberry Pi 2 Model B Pinout Diagram

In addition to the software normally needed to write UWP apps, I'll need to have some additional hardware. Naturally, I'll need to have a Raspberry Pi 2 Model B on which to deploy my solution. I'll also need a MicroSD card, an LED light, a 220 Ohm resistor, solderless breadboard, jumper wires, USB mouse and keyboard, and an HDMI monitor.

Raspberry Pi 2 Model B The Raspberry Pi 2 Model B is the computer onto which I'll deploy my UWP app. The Raspberry Pi 2 contains 40 pins (see **Figure 1**), some of which are General Purpose input/output (GPIO) pins. Using code, I'll be able to manipulate or read the state of these pins. Each pin has one of two values: high or low—high for higher voltage and low for lower voltage. This lets me turn the LED light on or off.

MicroSD Card The MicroSD card acts as the Raspberry Pi 2 hard drive. This is where the device will find its boot files and OS. It's also where the UWP app, once deployed, will reside. I could get away with SD cards as small as 4GB, but it's recommended to have 8GB. Naturally, the project requirements determine the size of the card needed. If, for example, I needed to store large amounts of sensor data locally before uploading, then I'd need a larger SD card to support a larger local file store.

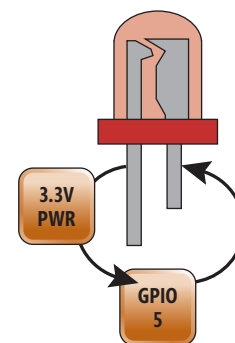


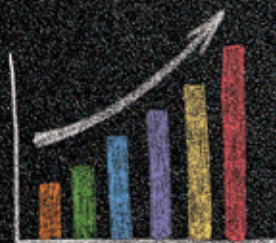
Figure 2 The Circuit Diagram

Spreadsheets Made Easy.



Fastest Calculations

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



Comprehensive Charting

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



Windows
Forms



Silverlight



WPF

Powerful Controls

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



Scalable Reporting

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

Download your free fully functional evaluation at SpreadsheetGear.com



SpreadsheetGear

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

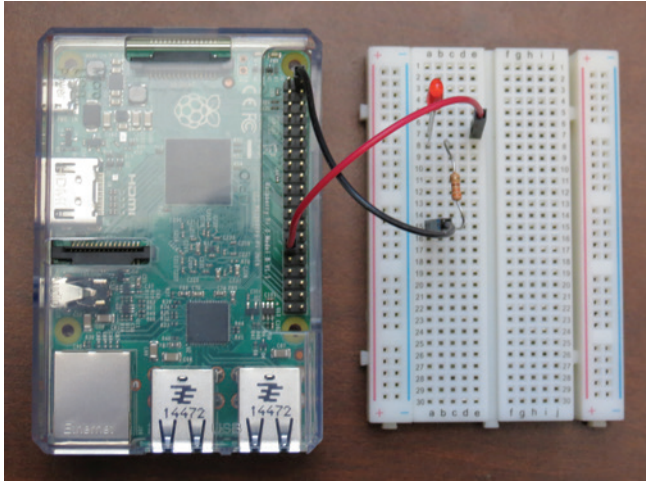


Figure 3 The Completed Wiring with Raspberry Pi 2 in a Clear Plastic Case

Solderless Breadboard and Jumper Wires In order to connect components to the Raspberry Pi 2, I'll need to create a path for electrons to follow from the Raspberry Pi 2 through my components and back to the Raspberry Pi 2. This is known as a circuit. While I could use any number of ways to connect the parts together, the fastest and easiest way is the solderless breadboard. As the name implies, I won't need to solder components together to create the circuit. I'll use jumper wires to make the connections. The type of solderless breadboard I use for this project has 30 rows and 10 columns of sockets. Note that the columns have two groupings of five: "a through e" and "f through j." Each hole is connected electrically to every other hole in its row and column group. The reason why will become apparent shortly.

LED Light and Resistor In this project, I'll connect the LED light to the Raspberry Pi 2 board. The pins on the Raspberry Pi 2 operate at 5 volts. The LED light, however, will burn out at this voltage. The resistor will reduce the extra energy to make the circuit safe for the LED light.

Ethernet Cable, USB Mouse and Keyboard, and HDMI Monitor The Raspberry Pi 2 Model B has four USB ports, an Ethernet jack and HDMI output, among other connectors. Once the UWP app is running on the device, I can interact with it very much the same way as if it were on a PC or tablet because I have a display and will be able to enter a ZIP code to pull down the forecast for a specific area.

Putting Windows onto the Raspberry Pi 2

To get started with Windows 10 IoT Core, I follow the directions at bit.ly/1025Vxl. The first step is to download the Windows 10 IoT Core Tools at bit.ly/1GBq9XR. The Windows 10 IoT Core Tools contain utilities, WindowsIoTImageHelper and WindowsIoTWatcher,

for working with IoT devices. WindowsIoTImageHelper provides a GUI to format an SD card with Windows IoT Core boot files. WindowsIoTWatcher is a utility that periodically scans the local network for Windows IoT Core devices. I'll be using them shortly.

Connecting the Hardware

In order to start creating a solution for the IoT, I need to make a "thing" with which to work. This is the part of an IoT project that many developers find the most intimidating. Most developers are accustomed to moving bits via code, not necessarily wiring parts together for electrons to travel around. To keep this simple, I take the very basic blinking LED light project (bit.ly/1025Vxl), but enhance it with real-time data from the Internet. The basic hardware supplies are the same: LED light, solderless breadboard, jumper cables and a 220 Ohm resistor.

In order to start creating a solution for IoT, I need to make a "thing" with which to work.

The Raspberry Pi 2 Model B has a number of GPIO pins. The state of many pins can be manipulated by code. However, some of these pins have reserved functions and can't be controlled by code. Fortunately, there are handy diagrams of the purpose of each pin. The diagram seen in **Figure 1** is known as a "pinout" and provides a map of the circuit board's interface.

Designing a Circuit

Basically, what I need to create is a circuit for electrons to flow through, as shown in **Figure 2**. The electrons start their journey

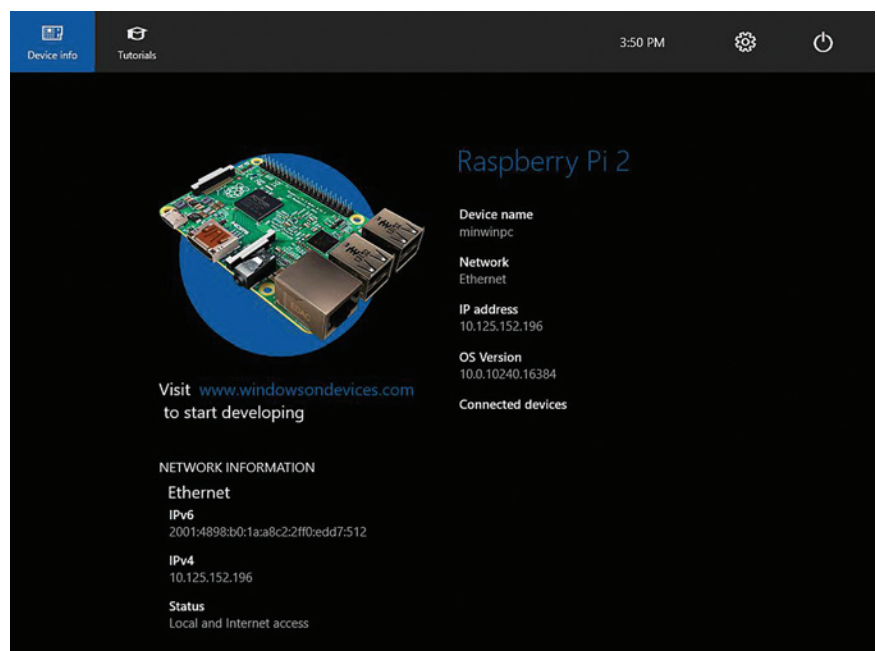


Figure 4 The Default Information Screen on Windows IoT Core for Raspberry Pi 2

FREE TRIAL

DocuViewware

Universal HTML5 Viewer & Document Management Kit

supports nearly 100 file formats



zero-footprint solution



super-easy integration



fast & crystal clear rendering



fully customizable UI
look & feel



mobile devices optimization



annotations, thumbnails
bookmarks & text search

HOT FEATURES IN THE NEW MAJOR VERSION

Office Open XML (.docx)
support

TWAIN acquisition

PDF form fields
interactions

Custom snap-ins

Higher speed and
lower memory usage

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



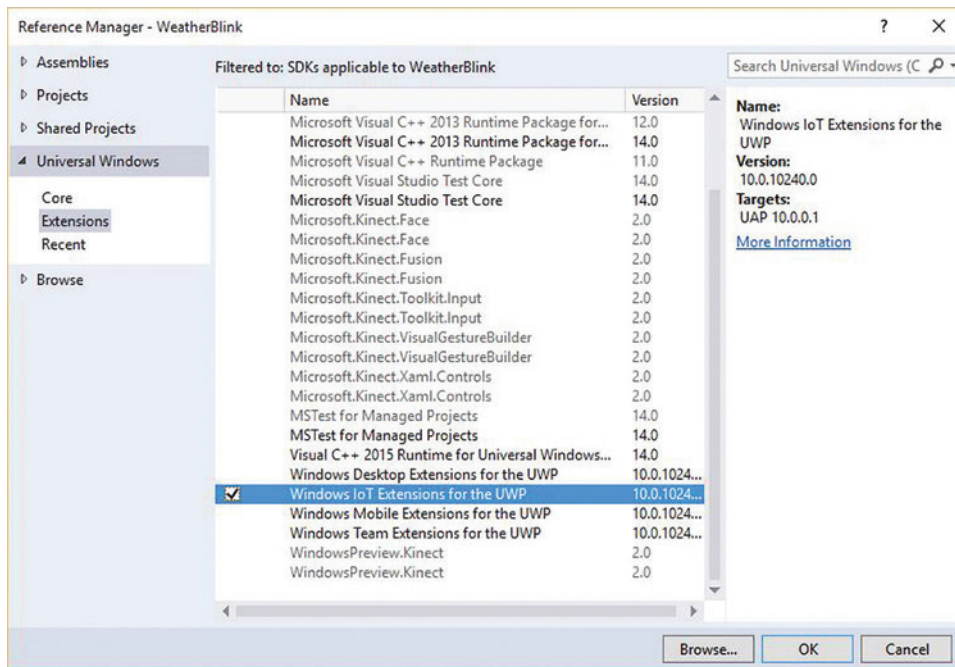


Figure 5 Adding a Reference to Windows IoT Extensions for the UWP in Visual Studio 2015

at pin 1, labeled 3.3V PWR in **Figure 1**. This pin supplies 3.3 volts of power to the circuit and it's this power that will light the LED. In fact, 3.3 volts is too much power for the LED light. To prevent it from burning out, I place a resistor on the circuit to absorb some of the electrical energy. Next on the circuit is GPIO 5, which, according to the pinout diagram, is physical pin 29. This pin, which can be controlled by code, makes the LED light "smart." I can set the output voltage of this pin to either high (3.3 volts) or low (0 volts) and the LED light will be either on or off, respectively.

Building a Circuit

Now, it's time to build the circuit shown in **Figure 2**. For this, I need to take the female end of one jumper cable and connect it to pin 29 on the Raspberry Pi 2. I then place the other end, the male end, into a slot on my solderless breadboard. I chose row 7, column e. Next, I take the LED light and place the shorter leg into the slot at row 7, column a, while placing the other, longer LED into the slot at row 8, column a. Now, I take the resistor and place one end into row 8, column c and the other into row 15, column c. Finally, I place the male end of the second jumper cable into the slot at row 15, column a, and connect the female end into pin 1 on the Raspberry Pi 2. Once all of this is done, I have something that looks like **Figure 3**.



Figure 6 The UI of the WeatherBlink UWP App

Booting up the Device

After I have Windows IoT Core installed onto a MicroSD card, I insert the SD card into the Raspberry Pi 2. Then, I connect a network cable, USB Mouse and HDMI monitor, and plug in the Raspberry Pi 2. The device will boot up and, eventually, the screen

shown in **Figure 4** will pop up (I make note of the device name and the IP address).

Writing the Software

With the hardware setup complete, I can now work on the software portion of my IoT project. Creating an IoT project in Visual Studio is easy. It's essentially the same as any other UWP project. As usual, I create my project by choosing File | New Project in Visual Studio 2015, and choosing Blank App (Universal Windows) as the template. I choose to call my project "WeatherBlink." Once the project loads, I'll need to add a reference to the Windows IoT Extensions for the UWP. I right-click on References in my solution in Solution Explorer and in the dialog box that follows, check the Windows

IoT Extensions for the UWP under Extensions in the Universal Windows tree (see **Figure 5**). Finally, I click OK.

Now that I have the correct reference added to my project, I'll add the following using statement to the top of the MainPage.xaml.cs file:

```
using Windows.Devices.Gpio;
```

The Windows.Devices.Gpio namespace contains all the functionality I need to access the GPIO pins on the Raspberry Pi 2. Setting the state of a given pin is easy. For example, the following code sets the value of pin 5 to High:

```
var gpioController = GpioController.Default();
```

```
gpioPin = gpioController.OpenPin(5);
gpioPin.Write(GpioPinValue.High);
```

Reading a pin's value is just as easy:

```
var currentPinValue = gpioPin.Read();
```

Because GPIO pins are resources that need to be shared across the app, it's easier to manage them via class-scoped variables:

```
private GpioPin gpioPin;
private GpioPinValue gpioPinValue;
```

And initialize them in a common method:

```
private void InitializeGPIO()
{
    var gpioController = GpioController.Default();

    gpioPin = gpioController.OpenPin(5);
    gpioPinValue = GpioPinValue.High;
    gpioPin.Write(gpioPinValue);
    gpioPin.SetDriveMode(GpioPinDriveMode.Output);
}
```

Creating the Simple UI

Because this is a UWP app, I have access to the full range of Windows 10 UWP interface controls. This means that my IoT can have a fully interactive interface with no additional effort on my part. Many IoT implementations are "headless," meaning that they have no UI.

PRECISELY PROGRAMMED FOR SPEED

DynamicPDF—Comprehensive PDF Solutions for .NET Developers

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



DynamicPDF

[WWW.DYNAMICPDF.COM](http://www.DynamicPDF.com)

TRY OUR PDF SOLUTIONS FREE TODAY!

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

ceTe software



Figure 7 Checking Weather and the Rate of Blinking

```
private async void LoadWeatherData()
{
    double minTempDouble = await GetMinTempForecast();

    // 38F/3.3C = 276.483 Kelvin
    if (minTempDouble <= 276.483)
    {
        Blink(500);
        txtStatus.Text = "Freeze Warning!"
    }
    else
    {
        Blink(2000);
        txtStatus.Text = "No freezing weather in forecast."
    }
}
```

This project will have a simple UI that'll display a message based on the weather forecast. If a keyboard and mouse are attached to the Raspberry Pi 2, end users will be able to enter a ZIP code and update the weather forecast information accordingly, as shown in **Figure 6**.

Making the Device Smart

In order to make my IoT device aware of the weather forecast, I need to pull down weather data from the Internet. Because this is a UWP app, I have all the libraries and tools accessible to me. I chose to get my weather data from openweathermap.org/api, which provides weather data for a given location in JSON format. All temperature results are given in Kelvin. **Figure 7** shows my code for checking the weather and changing the rate of blinking based on the results. Typically, frost warnings are issued once the air temperature gets to around 38 degrees Fahrenheit (3.3 degrees Celsius). If there's a chance of frost, I want the LED to blink fast to alert me that my garden is in imminent danger. Otherwise, I want the LED to blink slowly, to let me know that there is still power to the device. Because making a REST API call and parsing a JSON response in UWP is a well-covered topic, I've omitted that specific code for brevity.

The Blink method is straightforward—it sets the interval of a dispatch timer based on the parameter sent to it:

```
private void Blink(int interval)
{
    blinkingTimer = new DispatcherTimer();
    blinkingTimer.Interval =
        TimeSpan.FromMilliseconds(interval);
    blinkingTimer.Tick += BlinkingTimer_Tick;
}
```

The BlinkingTimer_Tick method is where the code to turn the LED on or off resides. It reads the state of the pin and then sets the state to its opposite value:

```
private void BlinkingTimer_Tick(
    object sender, object e)
{
    var currentPinValue = gpioPin.Read();

    if (currentPinValue == GpioPinValue.High)
    {
        gpioPin.Write(GpioPinValue.Low);
    }
    else
    {
        gpioPin.Write(GpioPinValue.High);
    }
}
```

The full source code is available at bit.ly/1PQyT12.

Deploying the App

Deploying the app to the Raspberry Pi 2 requires an initial setup on my PC. First, I'll need to change my architecture to ARM and then under the dropdown next to the play icon, I'll choose Remote Machine. The Remote Connections dialog (see **Figure 8**) appears, where I can either enter my device's IP address manually or select from a list of auto-detected devices. In either case, authentication doesn't need to be enabled. Last, I hit Select and now I can deploy my solution to the device.

Design Considerations

The world of IoT opens new opportunities and challenges for developers. When building an IoT device prototype, it's important to factor in the runtime environment where it'll be deployed. Will the device have ready access to power and networking? A home thermostat certainly will, but a weather station placed in a remote forest might not. Clearly, most of these challenges will dictate how I build my device, for example, adding a weatherproof container for outdoor scenarios. Will my solution be headless or require a UI? Some of these challenges will dictate how I would write code. For example, if my device transmits data over a 4G network then I need to factor in data transmission costs. I certainly would want to optimize the amount of data my device sends. As with any project that's purely software, keeping the end-user requirements in mind is critical.

Wrapping Up

While controlling an LED light from code might not change the world, there are many other applications that could. Instead of relying on a weather forecast API, I could connect a temperature sensor to the Raspberry Pi 2 and place it in or near my garden.

What about a device that could send an e-mail alert if it detected moisture in a particular part of my home? Imagine installing air quality sensors all over a major city or just in a neighborhood. Imagine placing weight sensors on roofs to determine if enough snow has fallen to determine if there's a risk of collapse. The possibilities are endless.

Go and build great things! ■

FRANK LA VIGNE is a technology evangelist on the Microsoft Technology and Civic Engagement team, where he helps users 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*. (youtube.com/FranksWorldTV).

THANKS to the following technical experts for reviewing this article: Rachel Appel, Robert Bernstein, Andrew Hernandez

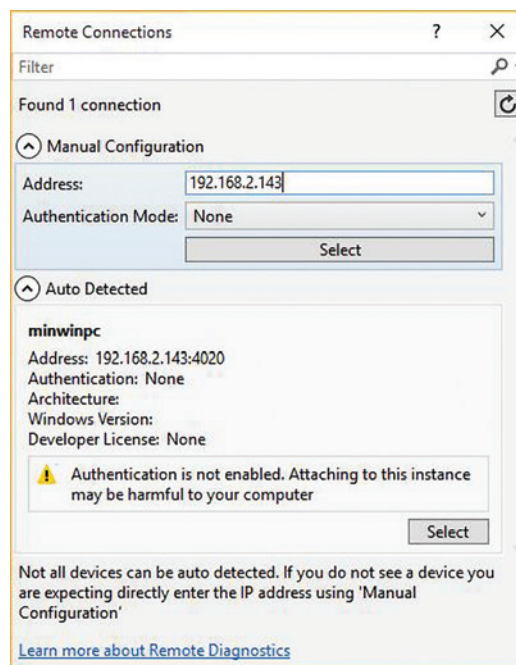
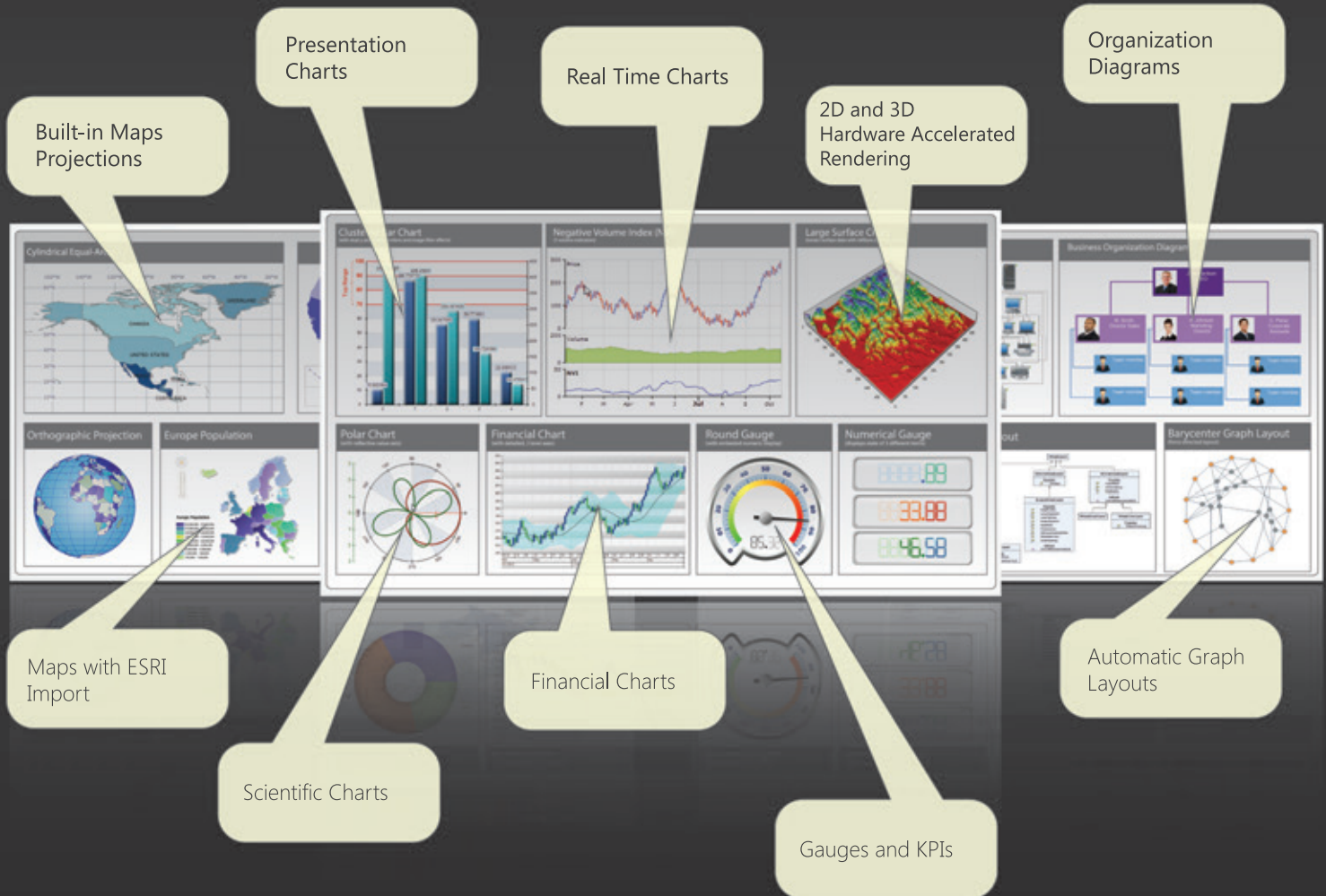


Figure 8 Remote Connections Dialog

Nevron Data Visualization

The leading data visualization components for desktop and web development in a single package.

NEW 2016 RELEASE COMING SOON!



solutions available for








Learn more at www.nevron.com today

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

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

GIVE YOUR

CAMPAIGN FOR CODE 2016 ★ VISUAL STUDIO LIVE!

Microsoft HQ

CODE A VOICE



Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS



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

SUPPORTED BY



Visual Studio
MAGAZINE

PRODUCED BY



MICROSOFT HQ • AUGUST 8-12, 2016

MICROSOFT HEADQUARTERS • REDMOND, WA



CONNECT WITH VISUAL STUDIO LIVE!



twitter.com/vslive – @VSLive



facebook.com – Search "VSLive"



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

TOPICS INCLUDE:

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

Have your code heard:
register to join us today.

**REGISTER NOW
AND SAVE \$400!**



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

USE PROMO CODE VSLRED2

VSLIVE.COM/REDMOND



Gods and Fools

Abraham Lincoln famously said, “You can fool all of the people some of the time, and some of the people all the time. But you can fool yourself any time.” (Or something like that. April Fool.) We geeks are especially good at the last of these. Before long, though, we won’t need to do it anymore.

I just finished watching the movie “Her,” starring Joaquin Phoenix as a lonely depressed geek and Scarlett Johansson as the voice of his AI bot girlfriend Samantha. She’s like Siri or Cortana or Alexa, on mental steroids. I wondered how a brilliant bot like her fell for a dweeb like him. And then it struck me: that’s precisely the point. She was built to bond to her user, because no one else would.

This idea of constructing an ideal partner, rather than finding one in the wild, occurs throughout human history. The earliest reference I can find is the ancient Cypriot sculptor Pygmalion. He sculpted a beautiful statue, then prayed to Aphrodite for a bride who would be “the living likeness of my ivory girl.” Aphrodite granted his wish by bringing the statue to life. Shakespeare used this idea in “A Winter’s Tale,” George Bernard Shaw’s play, “Pygmalion,” brought it to 20th century London, “My Fair Lady” set it to music and the Star Trek episode, “I, Mudd,” showed it backfiring. Robert Heinlein’s “Time Enough for Love” projects it 2,200 years into the future (about as far as the original Pygmalion was in the past): His self-aware computer character Minerva falls in love with a human.

But the work that most resembles “Her” is Lester Del Rey’s 1938 science fiction short story, “Helen O’Loy” (bit.ly/1naYFWp and downloadable as a PDF at bit.ly/1QnJ03h). Dave, a near-future robot repairman, invents artificial endocrine glands that provide emotional capabilities to robots—an early analog approach, rather than today’s digital. He and his partner Phil splice these into an off-the-shelf robot and, of course, she falls in love with him. They name her Helen for her beauty, and O’Loy for the alloys from which she is constructed. And she proceeds to light up their hitherto-empty lives.

How can human/bot love work? Obviously, physical bodies pose an obstacle. Minerva’s friends clone a human body for her from carefully selected genes, so she’s all set. In the obligatory lovemaking scene in “Her,” we hear suggestive audio but the screen is dark. Samantha later recruits a surrogate to perform the physical acts, but it doesn’t work out the way the characters want. Helen is built from rubber and metal, apparently quite well: “You know how perfectly I’m made to imitate a real woman ... in all ways. I couldn’t give him sons, but in every other way ...”

Normally a physical goddess like this wouldn’t pay attention to geeks like us, but we can program them to be attracted to the way we actually are: “Oh, you have such beautiful love handles.” What geek could resist?

We haven’t quite reached the point of Cortana evolving into Samantha. We still need to assist in our own deception, as I wrote last November (msdn.com/magazine/mt620019). But I foresee the day when humanity will no longer need to call on gods or fiction or even denial to attain the unattainable.

I foresee the day when humanity
will no longer need to call on
gods or fiction or even denial to
attain the unattainable.

We are the new gods, creating in our own image. Starting out crude, limited, buggy—and what’s more human than that? But constantly developing, improving; occasionally disrupting—and what’s more human than that, either? The mythologies we instantiate will echo forward from a strangely prescient past. Those ancient Greeks were onto something; it just took us a while to become able to build what they thought of.

The line between reality and virtuality gets blurrier by the day. At what point do they become indistinguishable? I wish I could just say “April Fool” and end this piece. But the seductiveness of our fool’s universe grows daily as technology progresses. How long until we refuse to leave it, thereby dooming the human species? As Phil says at the end of the story: “I’m an old man now and can view things more sanely; I should have married and raised a family, I suppose. But ... there was only one Helen O’Loy.” ■

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.

Get the Toolkit. Get the Team.



Whether you're developing an external application for a client or maintaining your company's internal legacy software, ComponentOne Studio's controls will speed up your development time and allow you to get to coding the fun stuff.

Solve business problems with controls that meet your users' every need.



Grids & Data Management

Get every function you need from a spreadsheet—without any heavy lifting, using our datagrid controls.



Data Visualization

Present large data sets out-of-the-box with our beautiful, flexible charts and gauges.



Reporting & Documentation

Migrate reports from another platform or generate your own with a full collection of controls.

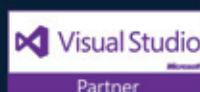


Scheduling

Offer instant Outlook-like functionality in any application.

WinForms | WPF | ASP.NET Web Forms | ASP.NET MVC | UWP | LightSwitch | Silverlight | ActiveX

Download your free trial at
www.ComponentOne.com



MORE THAN 50 CONTROLS

FOR XAMARIN.iOS, XAMARIN.ANDROID,
AND XAMARIN.FORMS



ESSENTIAL STUDIO FOR XAMARIN

- The most feature-rich chart and data grid components available.
- Create cross-platform mobile apps in C#.
- Design an attractive native UI for any device.
- Reuse your existing .NET code assets.

Visit us at
Xamarin Evolve
in Orlando, FL
April 24-28

Also available for free as part of the Syncfusion Community License!

Get started today!

www.syncfusion.com/msdnxamarin

 **Syncfusion®**