THE MICROSOFT JOURNAL FOR DEVELOPERS

FEBRUARY 2017 VOL 32 NO 2

msdn magazine

DevExpress

Your Next Great Desktop App Starts Here

From apps that replicate the look and feel of Microsoft Office® to high-powered data mining and decision support systems for your enterprise, DevExpress desktop controls will help you build your best, without limits or compromise.

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

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

FEBRUARY 2017 VOL 32 NO 2

msdn magazine

Azure Functions and App Service Architecture.....18, 26

COLUMNS

Microsoft

WE'RE CHANGING THE WAY YOU LOOK AT REPORTING

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

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

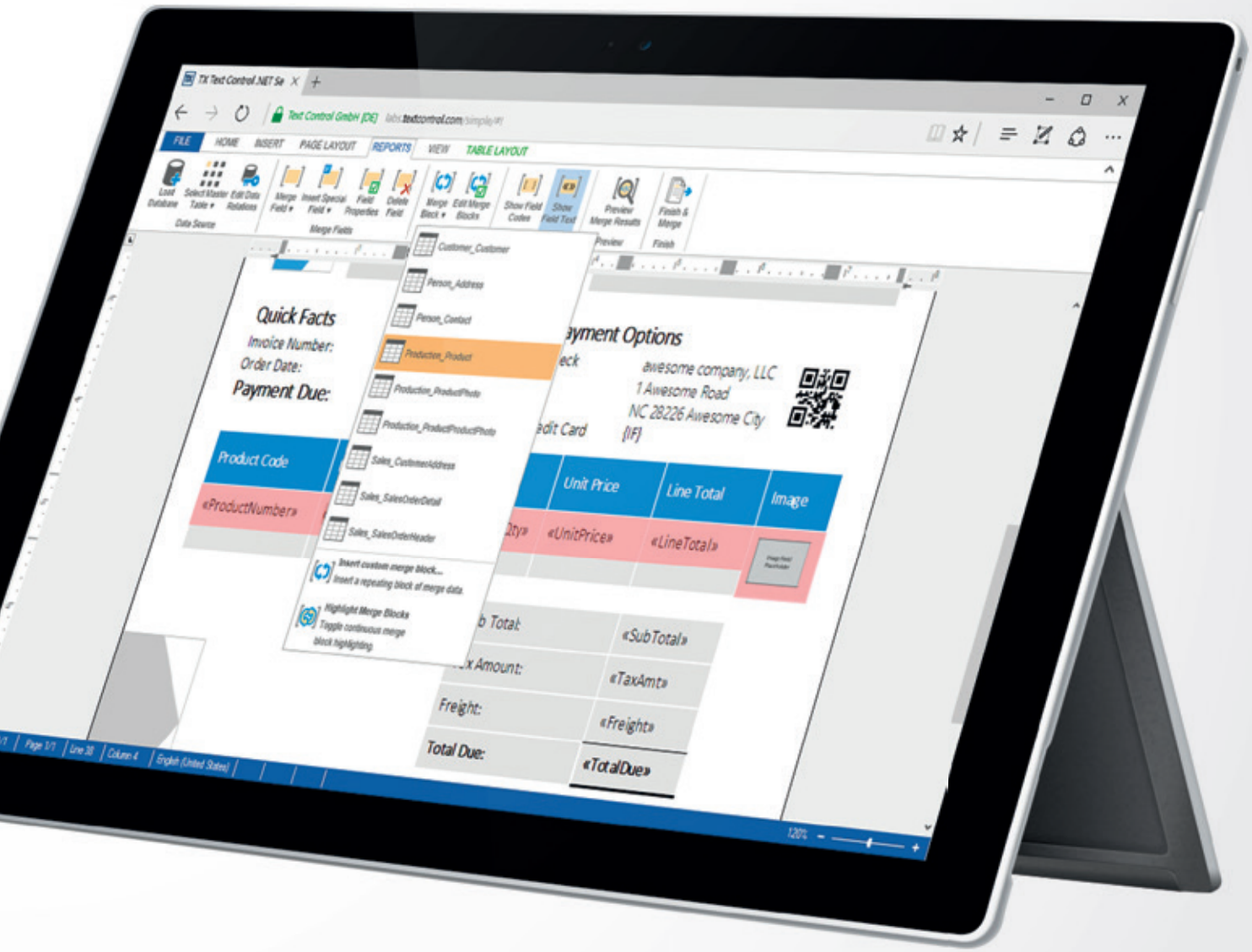
Follow the trend and switch to flow type layout reporting.

www.textcontrol.com www.reporting.cloud

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

Visual Studio Partner

REPORTING LIBRARIES FOR WINDOWS, WEB, MOBILE AND CLOUD APPLICATIONS



TEXT CONTROL

msdn magazine

FEBRUARY 2017 VOLUME 32 NUMBER 2

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

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

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

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

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

Reaching the Staff

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

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

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

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

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

Microsoft

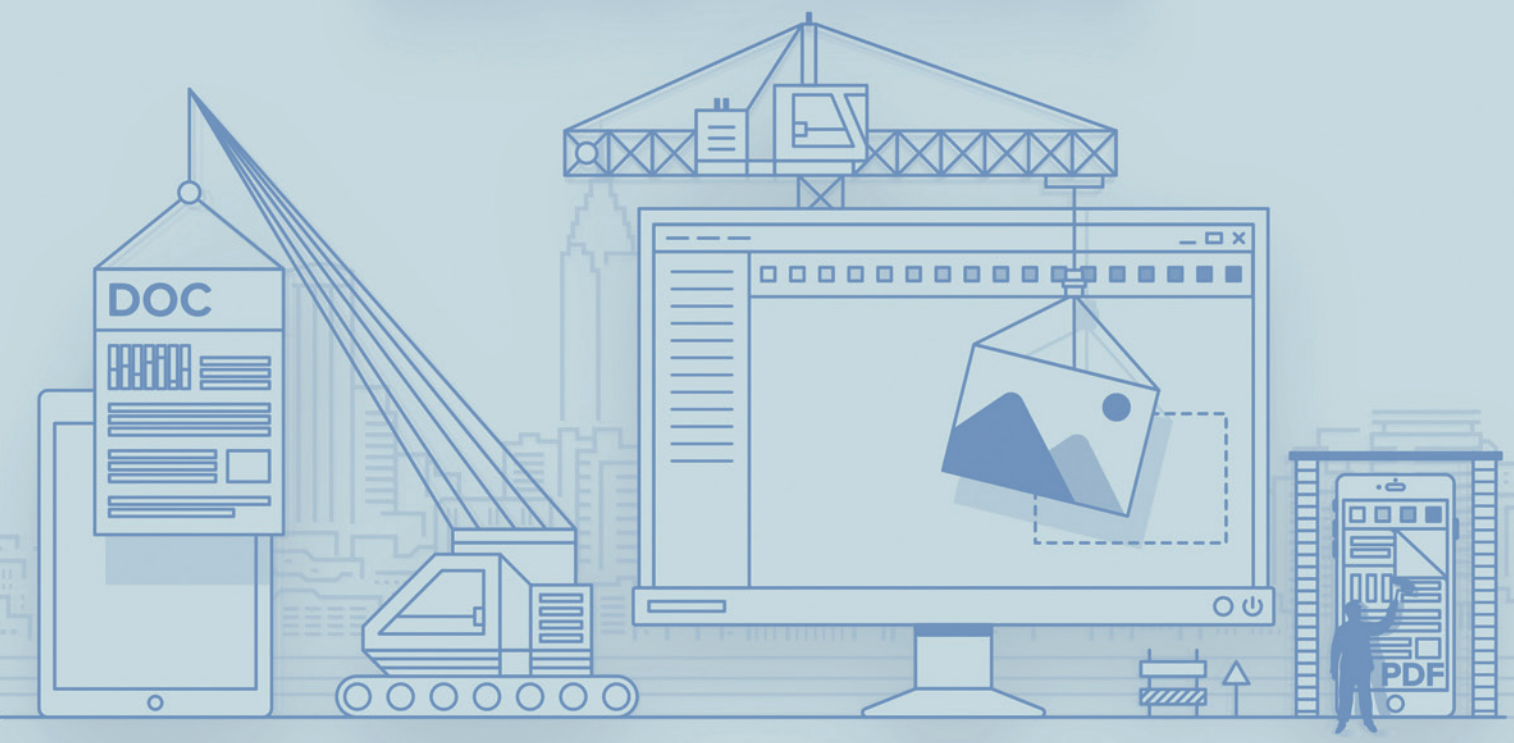
BPA WORLDWIDE

100 1906 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs



BUILD BETTER APPS WITH LEADTOOLS

DOCUMENT

- Document Viewer & Converter
- OCR, MICR, OMR & ICR
- 1D & 2D Barcode
- Forms Recognition
- PDF, DOCX, HTML, SVG, RTF, TXT
- Annotations & TWAIN

MEDICAL

- DICOM, CCOW & HL7
- PACS Client & Server Framework
- DICOMWeb (WADO)
- Web & Desktop Viewers
- Image Processing & Annotations
- Medical 3D (MPR, MIP, VRT)

MULTIMEDIA

- Play, Capture, Convert & DVR
- Media Streaming
- MPEG-2 TS, RTSP, HTML5 & more
- OGG, FLV, ISO, AVI, WebM & more
- Audio & Video Processing
- DirectShow & Media Foundation

.NET Windows API Java WinRT Linux iOS macOS Android JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1835

Editor's Note

MICHAEL DESMOND

Our Serverless Future

This month's issue leads off with a pair of articles focused on new elements within Microsoft Azure and the impacts they have on software development. Yochay Kiriaty and Stefan Schackow explore the new Azure App Service architecture, which provides a rich Platform as a Service (PaaS) for Web, mobile and API applications. App Service is great for enabling apps that can run at global scale, by providing underlying infrastructure and managed services that let developers focus on writing great application code.

The other Azure-oriented feature this month, written by Joseph Fultz and Darren Brust, is "Serverless Architecture with Azure Functions." As Fultz explains, the term serverless architecture describes a design in which there is no server infrastructure to manage by the application team.

> "Ideally, the app team wouldn't even have to think about scale as the serverless compute and third-party services should automatically handle scale and availability."
>
> *Joseph Fultz, Cloud Solution Architect, Microsoft*

"This means that all the moving parts of the system are either third-party services like Azure SQL Database, DocumentDB, EventHubs or Office365, which are also known as Back-end as a Service (BaaS)," Fultz says. "Or they are custom code hosted in a system-managed container to provide the serverless compute, which is also referred to as Functions as a Service (FaaS)."

The approach reflects the proliferation of mature, third-party services in the cloud era, as dev organizations shift on-premises compute infrastructure to services in the cloud. Fultz points out that not so long ago hosting virtual machines was of huge value to IT organizations, yet today many of those virtualized activities have shifted to a service consumption model. In short, it's become more burdensome to roll your own than it is to have someone else roll it for you.

"Ideally, the app team wouldn't even have to think about scale as the serverless compute and third-party services should automatically handle scale and availability," Fultz says.

The upside is reduced complexity and cost, which enables organizations to focus more resources on adding value to the business rather than building and managing infrastructure. The downside? Less control of the implementation of the environments that host application services.

So how should developers proceed as they consider serverless approaches? Fultz says the most important thing is to understand the services providers you're coupling your applications to and how those commitments impact your DevOps pipeline.

"Developers must be familiar with the intricacies of deploying their various environments, such as dev, test and production, against a matching set of services for the target environment," Fultz says, adding that familiarity with logging and troubleshooting capabilities is also important.

"Without access to the server directly, one has to rely on what is provided by the host of the service being used," he continues. "Sometimes there is rich integration with management and monitoring, like in the case of Azure SQL Database, but in other cases there may not, as in the case of Azure Functions, which is still new to the scene."

As for how Azure Functions relates to Azure App Service, which Kiriaty and Schackow write about this month, Fultz describes App Service as the PaaS on which Azure Functions is built. App Service works behind the scenes of Azure Functions, handling a range of activities from the simple, like storage of configuration settings, to the complex, such as scaling operations. As Fultz describes it, Azure Functions is essentially "App Service plus the Web Jobs SDK plus Azure Functions-specific implementation bits."

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

Switch to Amyuni PDF

AMYUNI

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

NEW v5.5

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

Complete Suite of Accurate PDF Components

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

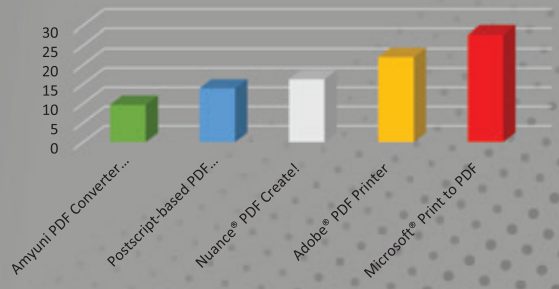
Other Developer Components from Amyuni®

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

High Performance PDF Printer for Desktops and Servers

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

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



CERTIFIED FOR Windows Server 2008

Windows Server 2012 Certified

Windows 7

Windows

AMYUNI Technologies

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

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

All development tools available at

www.amyuni.com

Age Before Beauty: Success and the Older Developer

I often counsel older software engineers who fear they are "too old" for the tech industry. I've had hiring managers tell clients that they're "too senior" or "over-qualified" or "too expensive." And I've had companies say outright that they're looking for "impressionable developers" or "software engineers under age 25."

In the technology industry, with every additional year of experience, it's easy to feel that your employability diminishes. I've found a few tricks that have been effective for software engineers.

Excessive Experience

There's a myth that one must list all of one's experience on a résumé or LinkedIn profile. This is unnecessary. A résumé is a marketing snapshot of experience, not a biographical record to be preserved by historians. And as a snapshot, it should emphasize your fit for the company to which you're talking.

So, if you're 45 years old, leave out your experience until age 30 or 35. Instead, showcase the last 10 to 15 years. If you do get asked to describe your earlier experience, share the truth briefly. An interviewer may ask just out of curiosity, not to "trap" or "expose" you.

Timing Is Everything

If you're showcasing only your most recent work experience, leave the dates off your education altogether. For example: If you're listing work experience from only 2006 to 2016, it's confusing to show a college graduation date of 1985. It makes people wonder what you were doing during the 21 years in between! My advice in such a case: Leave off college graduation and attendance dates.

Mind the Gap

You don't need to mention every job that you've ever had. Say you went through a rough patch of underemployment and you worked at a retailer for a few months—it's fine to leave that job off your résumé. In fact, doing so makes your résumé more focused, eases your interview prep and simplifies your story for the recruiter.

If you feel insecure about, say, a six-month gap on your résumé (you need not), just leave those months out on the résumé. So, instead of saying Job A went from "December 2014 to January 2016" and job B went from "August 2016 to Present," it's OK to say that job A was "2014 to 2016" and job B was "2016 to Present." If you do this, be consistent and remove months from all other dates and date ranges in the résumé.

Formidable Seniority

To avoid the "over-qualified" label, downplay your job titles so you don't seem too senior for the opening. For example, if you've been a Principal Software Engineer and you're applying for a "Software Engineer" position at a tiny start-up, just list "Software Engineer" on your résumé.

Also, if you manage a division of 60 developers and now you're interviewing for a smaller company, describe your job as "managed a staff of senior software engineers." Don't say "managed an annual P&L of $10 million for a 60-developer business unit."

And here's a tip: If you're trying to decide between wording something in a way that gets you a job (but feels wrong) and something that may not get you a job (but feels more right), choose what feels more right. Always err on the side of truth.

Misplaced Style

My older clients unnecessarily age themselves with how they dress. I highly recommend working with a fashion stylist to have a wardrobe that's chic and bold. It may be one of your best investments.

This doesn't mean that you start dressing like Justin Bieber. What's important is that you think carefully about how you dress, feel authentic in what you wear, get others' feedback on how you come off, and are careful to avoid looking "too old" or "too stiff."

Say Less

I often tell my older clients that their biggest concern isn't the hiring manager; it's their own loose lips. Only "talk shop" about technical topics—and even then, do so with reticence. Ageists may unconsciously interpret talkativeness in older developers as pre-senility, but in younger developers as enthusiasm!

Also, avoid talking or asking about "family" concerns—insurance, benefits, vacations—in interviews. It's easy to get a sense of how lifestyle-friendly a company is without being direct. Those questions are fair game once you get an offer in writing.

Plan to Win

For older developers, competing with 21-year-old engineers can be a losing battle. Experience and "wisdom" only get you so far.

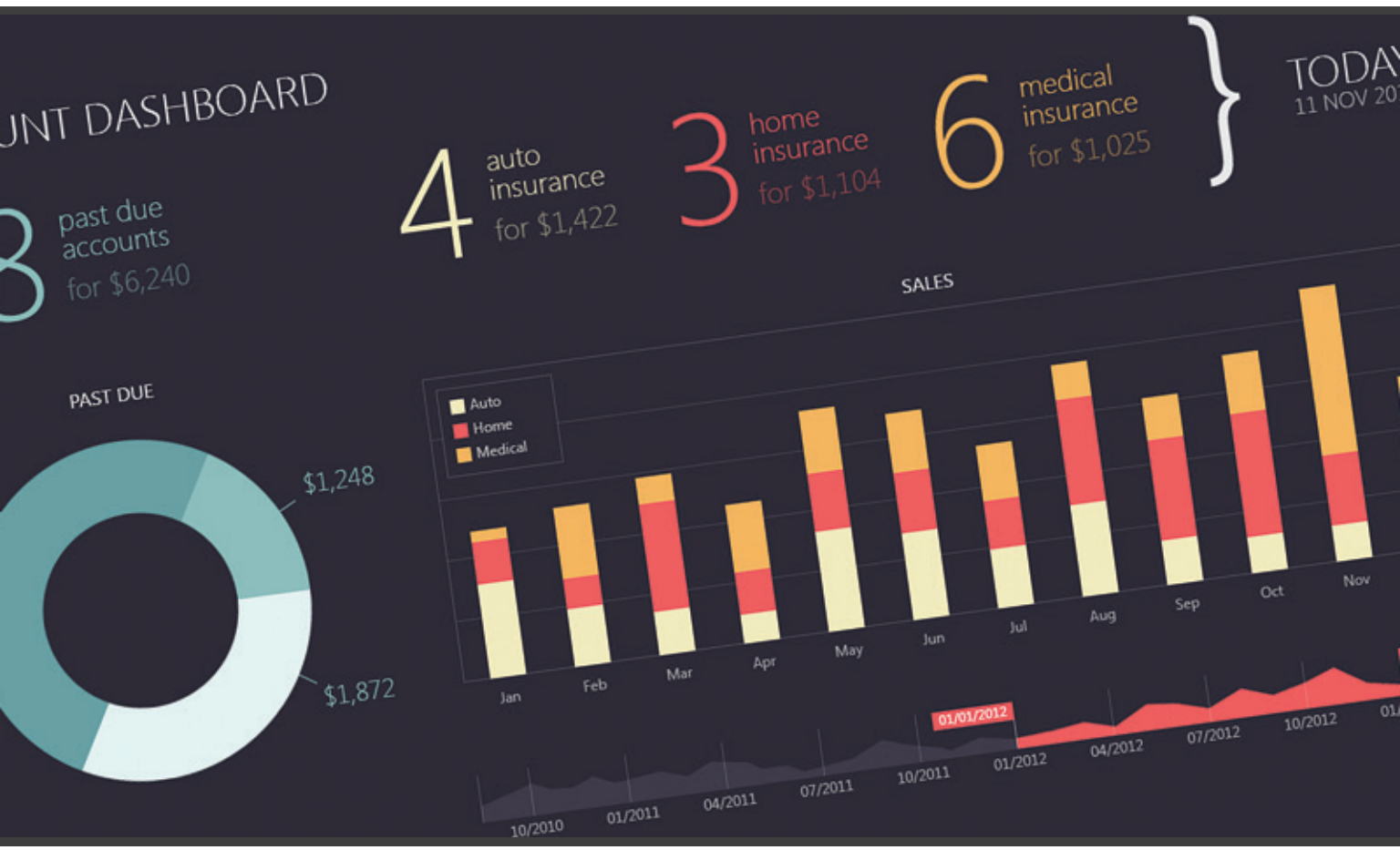
In this industry, especially for individual contributor roles, the key is to double down on sharpness. Do this with *over*-preparation for interviews, *over*-familiarity with the latest frameworks and tools used by the company, *over*-researching the company and its products, *over*-crafting compelling anecdotes for behavioral interview questions, and *over*-cultivating physical and mental vitality through exercise and meditation. ■

KRISHNAN RANGACHARI *is a career coach for software engineers. Visit RadicalShifts.com for his free career masterclass. Visit ByteshiftResumes.com for his résumé reinventions.*

DevExpress®

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

DINO ESPOSITO

In-House Business Intelligence with Events and CQRS

I was discussing a few wireframes with a customer recently to verify the effectiveness of a business process and overall quality of the UX the team was building. In the system were a few shared items of work that users could pick up and start processing. The customer was an overall technical person, but all she could envision was a relational database table with a record indicating who picked up what and the current state of processing.

Needless to say, a similar architecture would definitely do the job. With a different and event-based architecture, though, you can grab a lot more information about what's going on in the system. The core question, therefore, is the following: Might your customer need—now or in the near future—to access and consume the extra information you can get out of events?

In this column, I'll pick up where I left off in last month's column and add a business intelligence (BI) module to the sample application dealing with meeting rooms to book. The term BI these days is quite an overloaded buzzword often associated with specific technologies and products. At its core, though, BI refers to a set of techniques to grab and transform raw data into useful information available to analysts to improve processes or to use as mere statistical evidence.

By adding event sourcing to an application, you get a hold of raw data. By combining event sourcing and CQRS you end up with raw business events stored in the command stack properly denormalized for the sake of the application core functions. At any time, though, you can add an extra module that reads raw data and transforms that into other meaningful chunks of information for whatever business purpose you might have.

Never Miss a Thing

When it comes to highlighting the benefits of event sourcing, the first point usually mentioned is this: "With events you never miss a thing of what happens within the system." It couldn't be truer, but it probably deserves a bit more of a pragmatic explanation. Have a look at **Figure 1**.

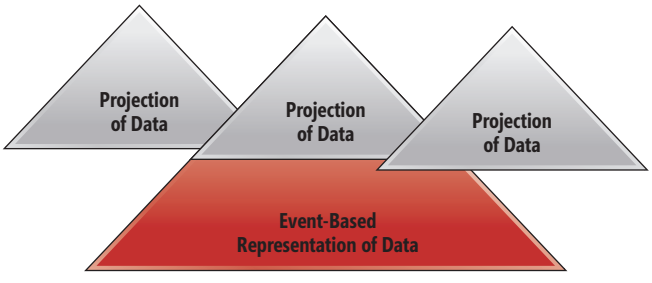


Figure 1 **Multiple Projections Can Be Built on Top of Raw Events**

In a create, read, update, delete (CRUD) system, you typically have one representation of data—mostly relational—and one or more simple projections that most of the time just adapt tabular data to the needs of the presentation layer. With event sourcing, you take this model much further, and lowering the abstraction level of the stored data is the key factor. The more domain-accurate information you store, the richer and more numerous projections you can build at any later time.

> At its core BI refers to a set of techniques to grab and transform raw data into useful information available to analysts.

In a software system, user actions are the smallest piece of observable behavior, and business events caused by those actions are the most basic piece of information you can store. In my previous column (msdn.com/magazine/mt790196), I used the MementoFX framework via NuGet to transparently handle and persist relevant events in the lifetime of domain aggregate objects. I also used special synchronization tools—called denormalizers—to create a viewable projection after each relevant business event. In the end, all the column showed was rewriting a CRUD system according to the Event-Command-Saga (ECS) pattern. Let's see what it takes now to add another projection to fill out the dashboard screen of a manager.

Toward Your Own BI Layer

In **Figure 2**, you see the primary screen of the sample application. As mentioned, it's a booking system for a shared resource like a set of meeting rooms.

As you can guess from the figure, any users allowed into the system have the chance to book a room and then move or cancel the reservation. In a classic CRUD-oriented system there's a Booking table where each record identifies a reservation. When a reservation is moved, starting time and length are overwritten and when a reservation is canceled, the record is simply deleted. At any time, a plain query of records returns the current state of

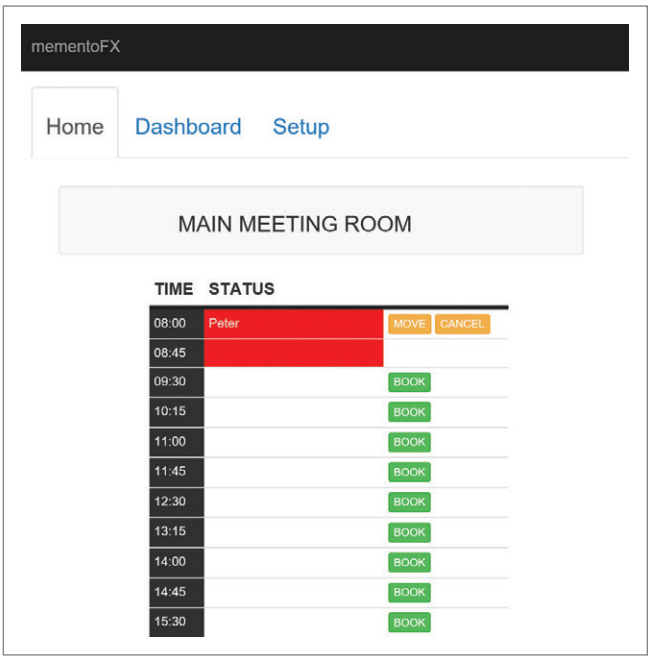


Figure 2 **The UI of the Sample Booking System**

the bookings. If you turn the plain CRUD into a historical CRUD (see the May and June 2016 installments of this column at msdn.com/magazine/mt703431 and msdn.com/magazine/mt707524, respectively), you can track a fixed number of events and have them saved with any relevant information in some other tables. In which way, then, is an event-sourcing, MementoFX-based solution preferable over a plain-and-simple historical CRUD? It's all about code flexibility and resiliency of the final software artifact.

With MementoFX, you focus on the relevant domain behavior and events. You model domain objects around these needs and let it go. In return, you have an API to query what happened, the state of the system at a given date and time, and whether you have a log of all events or just raw data aggregated and transformed in some custom way that makes sense for some occasional business purpose. This is the raw, intended meaning and essence of what people call BI.

Building a Plain Log of Events

With reference to the sample application, you have a RavenDB database that stores all raw events and a SQL Server denormalized table that contains the list of pending bookings. Stored events comprehend the event that created a booking and all successive events that moved that booking to a different time, and even events that canceled that booking. A canceled event doesn't show up anymore in the main UI and so it's for intermediate slots to which a booking was moved. All of these events aren't relevant for building the primary UI, but they're crucial for building a dashboard UI for a manager or an administrator.

As an example, let's see what it takes to group all bookings in the system (or bookings created in a given interval of time) and show the entire history of each, as shown in **Figure 3**.

The system counts 16 pending bookings, but each booking consists of one or more events. For example, the highlighted booking was first created and then moved twice to different slots on different days.

To produce a similar screen, you must definitely query all events in the event store and then work them out to reach a shape that suits the intended UI. The Razor code that produced the actual view received a data model like this:

```
public class BookingWithHistory
{
  public BookingWithHistory()
  {
    History = new List<BookingHistory>();
  }

  public BookingSummary Current { get; set; }
  public IList<BookingHistory> History { get; set; }
}
```

The BookingSummary class represents the current state of a given booking and is the class behind the primary view of **Figure 1**.

```
public class BookingSummary : Dto
{
  public Guid BookingId { get; set; }
  public string DisplayName { get; set; }
  public DateTime Day { get; set; }
  public int StartHour { get; set; }
  public int StartMins { get; set; }
  public int NumberOfSlots { get; set; }
  public BookingReason Reason { get; set; }
}
```

An instance of this class is created during the denormalization process after each "create" or "move" action. This class is persisted and read via Entity Framework to and from a classic SQL Server database. In other words, this class is the item that forms the default view of the system around the current and up-to-date snapshot of the state. Technically, this class belongs to the read model.

The view you get in **Figure 3** instead collects data also from raw events logged in the event store, a RavenDB store in the example. The following code snippet shows the queries ran against the event store to get all events of interest that happened in the lifetime of the booking with the given ID:

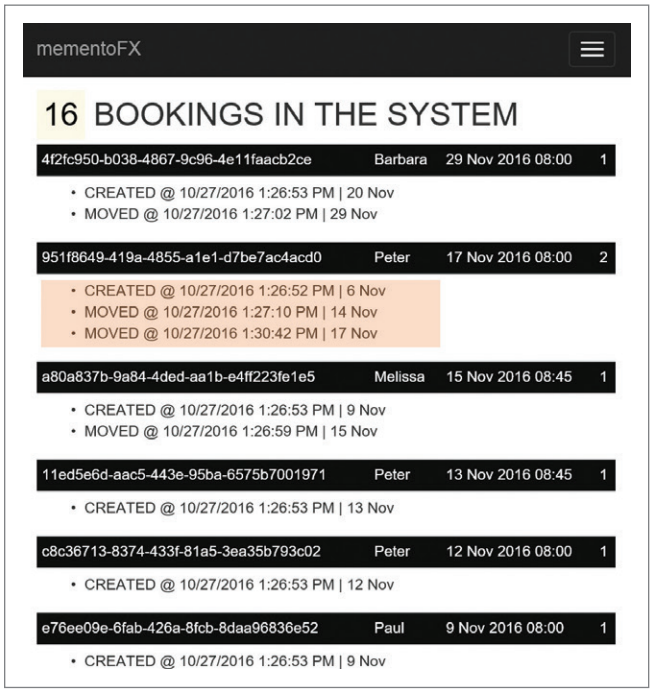


Figure 3 **The Full Log of Events in the System**

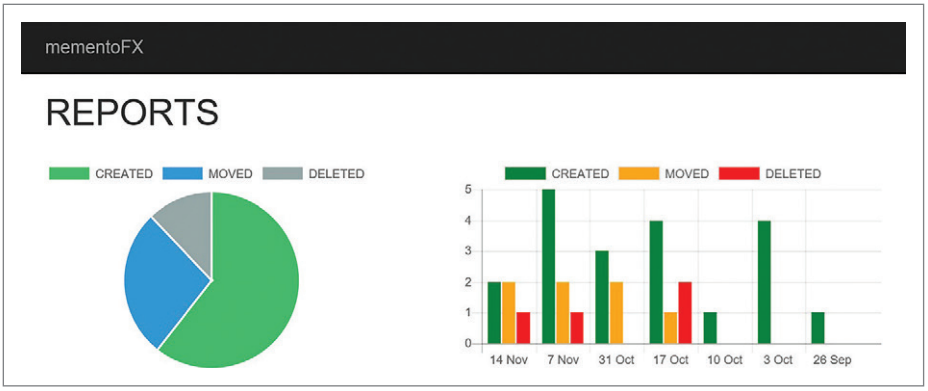


Figure 4 **A Graphical Report of Booking Actions in a Given Time Interval**

```
var createdEvents = EventStore.Find<NewBookingCreatedEvent>(e =>
  e.BookingId == bookingId).ToList();
var movedEvents = EventStore.Find<BookingMovedEvent>(e =>
  e.BookingId == bookingId).ToList();
var deletedEvents = EventStore.Find<BookingCanceledEvent>(e =>
  e.BookingId == bookingId).ToList();
```

The union of those events delivers the full history of a given aggregate. Replaying all those events—namely, applying sequentially the state of each event to a fresh instance of the aggregate object—returns the current state of the object for display or processing purposes. It goes without saying, though, that you can add a date filter to the query on events and thus rebuild the state of the domain object—the booking—up to a given date.

Extrapolating Some Business Information

Let's say you're a manager responsible for internal processes. In your role, you want shared resources, such as a meeting room, to be used effectively. How can you verify that and update booking policies accordingly? **Figure 4** provides useful information to answer that question.

The pie chart shows how many bookings were created in the time interval and how many of them have been later moved or canceled. The bar chart, instead, breaks up the same information on a weekly basis. At a first rough analysis, it seems that nearly half of reservations are moved at some point and about one of every four is even canceled.

If you're a manager willing to improve processes, the graphs in **Figure 4** might sound an alarm. Whatever way you look at it, the number of changes after reservations are made is significant. Is this preventing other potential users to book their rooms with ease? To stop the alarm from ringing, you might want to look at the average coverage of each room in the same time interval. If this particular coverage graph isn't in your dashboard already, having developers add it doesn't really require long hours of work. In the end, it's just about writing another predicate for a LINQ-style query:

```
var eventsByWeek = bookingStatuses.GroupBy(b => b.WeekDate).ToList();
foreach (var weekEvents in eventsByWeek)
{
  var wr = new WeekActivityReport
  {
    StartOfWeek = weekEvents.Key,
    TotalBookingCreated = weekEvents.Count(),
    TotalBookingMoved = weekEvents.Count(b => b.Moved),
    TotalBookingDeleted = weekEvents.Count(b => b.Deleted)
  };
  reports.Add(wr);
}
```

In particular, the sample application gets all events for all bookings in the given time interval and groups them by week. Next, it creates a weekly activity report object that is then easy to pass to ChartJS for creating dazzling graphics.

The most important aspect of event sourcing that I meant to emphasize in this column is that once you have raw information saved at the lowest possible level of abstraction, you can use it in many different ways at many different times. Sticking to the meeting rooms demo, you can deploy the manager dashboard in a successive release or make it another brand-new product. You can also parse all events in the lifetime of the application and build new data projections on top of it for whatever business goals you can think of. More than everything else, though, any successive development effort is for the most part independent from what you have and building it doesn't affect what you have in place already. That's the ultimate return on any investments you make on CQRS and event sourcing combined.

Wrapping Up

Processing business events in software is nothing new. You can probably achieve similar results using a historical CRUD (H-CRUD)—just a fancy name for any sort of handcrafted solution that lets you track all the different states of a business object. Event sourcing does the same job, except that it operates at a different level of abstraction and relies on more powerful and tailor-made tools (such as event stores) and patterns (such as event sourcing) in the context of specialized architectures (such as CQRS).

I've been writing about CQRS and event sourcing in *MSDN Magazine* for quite some time now and in a way I formed the idea that most people agree on the relevance of CQRS and events, but find it hard to find a place in which to start. To these people I recommend getting back to H-CRUD, which I wrote about back in May (msdn.com/magazine/mt703431) and June (msdn.com/magazine/mt707524), and then back to more recent articles about the ECS pattern (also known as CQRS/ES) and MementoFX. That should help you start from a familiar mindset and then progressively enhance it up to reaching a point in which you're doing old things in a new and more powerful way.

All this said, software is not about magic or religion. Software is about getting things done, preferably in a way that works for the customer and for the development team. Along with event stores, buses, event patterns, and architectures, MementoFX helps in having things done quickly and effectively. ■

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: *Andrea Saltarello*

Data Quality Made Easy.
Your Data, Your Way.

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

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

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

Microsoft SQL Server Integration Services | Microsoft Dynamics CRM | Microsoft .NET | Microsoft Azure | Microsoft Excel

Start Your Free Trial

www.MelissaData.com/msft-pd

Germany
www.MelissaData.de

United Kingdom
www.MelissaData.co.uk

India
www.MelissaData.in

Australia
www.MelissaData.com.au

MELISSA DATA®
Your Partner in Global Data Quality

www.MelissaData.com | 1-800-MELISSA

JULIE LERMAN

First Look at Azure Search—A Handheld Walk-Through

I've been curious about Azure Search since I first heard about it from Pablo Castro, who I knew as one of the originators of Entity Framework and OData. He's currently a director of engineering at Microsoft and has always been a true innovator and big-time data geek. His Medium article on how Azure Search came about as a startup project at Microsoft is a fascinating read (bit.ly/2gzVFTQ).

Azure Search is a service that lets you use Microsoft's processing and intelligence to add sophisticated search capabilities to your own data. The service builds its own search indexes from your data, keeping its own "copy" of your data on which it can perform its searches. You can automate Azure Search's process of updating the indexes as your data changes and this is even easier if your data source is an Azure data store such as DocumentDB.

I found it easier to understand the basics of Azure Search by diving in—so that's what I'll do in this article, rather than just trying to explain. You can create the service and indexes in code using REST or other APIs such as the .NET Client. But you can also set things up visually in the Azure Portal, which is where I'll begin—creating and querying a free search service in the Azure Portal with a sample data store. I'll handhold you through this first exploration and share with you some of my discoveries. With the comprehension that this first step provides, you'll be prepared to dig further into the services capabilities and begin interacting with it using the REST API or the .NET Client API.

If you have an MSDN subscription, you already have an Azure subscription you can use to follow along. We'll use a free service so as to not eat up your credits. If you don't currently have an Azure subscription, you can quickly set up a free trial at azure.com/free.

Create a New Search Service

After choosing to add a new resource in the Azure Portal, you'll see Search within the "Web + Mobile" category, although I find the easiest path is to just type search in the search box. (That's awfully meta, isn't it?) Setting the service up requires that you provide a service name, which becomes the first part of its URL. I'll use thedatafarm, which gives me the URL thedatafarm.search.windows.net. As with any resource, you have to choose to add this into an existing resource group or to create a new one. I don't have anything else related to my service yet, so I created a new resource group that I named datafarmsearch-group in East US. The last bit of info for setting up the resource is to select a pricing tier. Azure Search has a free pricing tier that's great for testing out the service. It allows for 10,000 documents, three indexes, and 50MB of storage with no scaling. You're allowed to set up one free Search Service in your subscription. Be sure you click the Select button after choosing the Free tier or you'll get the default Standard. Guess how I found out about that mistake.

When you click Create, Azure will verify your settings and if everything checks out, it should take only a few seconds for the service to be live.

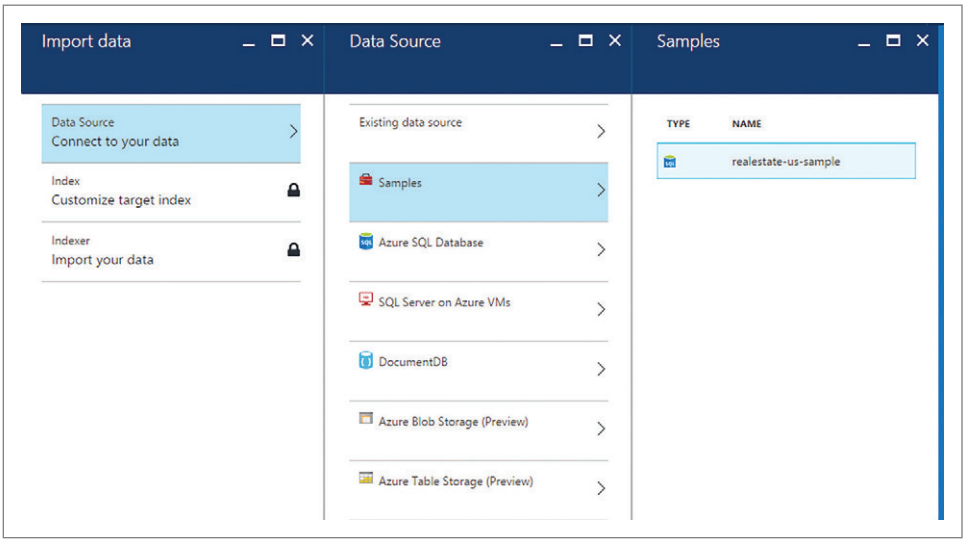


Figure 1 **Azure Search Already Knows How to Connect to Data Stored in Azure**

So now you have a service, but what will you search? On my first go-round with Azure Search I spent a lot of time looking for a search-worthy data set and then figuring out how to share that data with my service. It was fun, but in hindsight, I wish I'd begun with one of the sample data sets that Azure Search provides just to get my feet wet. So that's what I'll do here.

Search is performed not on data, but on indexes of data, and the indexes are in the form of documents. For us relational database people, an index is roughly like a table while a document is akin to an individual row in a table—a single

File Format APIs

Working with Files?

CREATE CONVERT PRINT

MODIFY COMBINE

FREE TRIAL

Aspose.Total

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

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

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

ASPOSE
File Format APIs

CONTACT US

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

sales@asposeptyltd.com

Try for FREE at
www.aspose.com

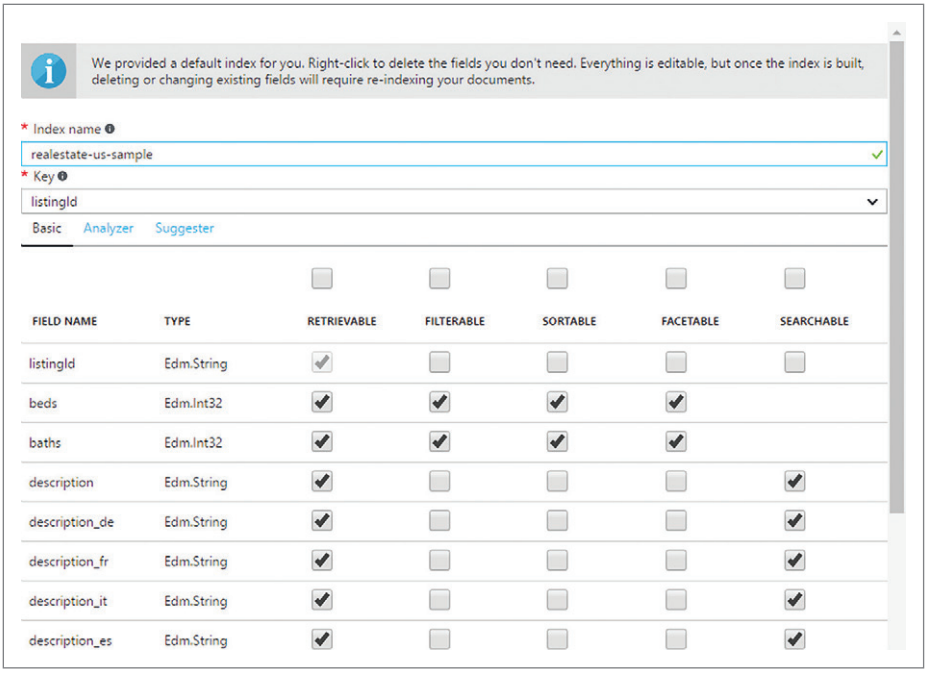


Figure 2 **First Few Rows of the Default Index Created from the realestate-us-sample Data**

unit of data. So the first thing to do is create an index of the data you want to search. The data can come from a variety of resources. Creating indexes from data that lives in another Azure service is the quickest path, although this is simplest to do with services in the same subscription as the Search. You can point to an existing Azure DocumentDB, an Azure SQL database or a SQL Server that lives on an Azure VM. As I'm writing this, Azure Table Storage and Blob Storage support are currently in preview. Azure Search gives you access to some pre-created sample data. So rather than begin by clicking the Add index option, choose Import data, which will also create the first index for you.

In the Import data blade, shown in **Figure 1**, choose Samples and then the realestate-us-sample data. Today as I write this article, that's the only option, but there are more samples coming. You can see by its icon that this is an Azure SQL Database, so you'll be creating an index of documents from the relational data. Search will transform the data into the structure it needs for indexing. It's a three-step process. First, you select the data source, then you define the index and, finally, you import the data into your index.

Define an Index

After you select the sample, the portal will present you with a grid listing all of the fields it discovered in the sample data source. This is the step where you define your index and, in this case, the service's wizard has pre-defined the index for you (**Figure 2**). This is your only opportunity to fine-tune the index before importing the data. Redefining an index means re-importing your data, which may not be desirable in a production environment.

An index against Azure SQL Database or SQL Server can target only a single table or view. If you're using a non-relational data store as your source, Search comprehends the full graphs that are stored within a particular document or blob. So what you see here is from the single table in the database. There are 26 fields, and the grid lets you define how each field will be involved in the search, using the options Retrievable, Filterable, Sortable, Facetable and Searchable. So, as you can already see, search provides some flexibility beyond just typing in a search term. Across the grid are columns to define how the field will be used in the search. The index wizard selected some defaults for you.

You might not need to be able to search on all of the fields. For example, there are seven description fields, one in English and each of the others in a different language. If the application that's using this resource will be used only by people in Quebec, and you plan to support only French and English searches, you could delete the other five fields from the search index. In the portal, you do this by right-clicking on the field and choosing Delete.

Defining indexes is something that takes some thought and planning, but I'll just go with the selections the importing tool made. But do take some time to look over the grid to understand what's being defined. Notice, for example, that the description fields are searchable and retrievable, but not filterable or sortable. In contrast, simple, scalar fields like square feet (sqft) and price are both sortable and filterable.

> For us relational database people, an index is roughly like a table while a document is akin to an individual row in a table—a single unit of data.

Using an Indexer to Import the Data

Now that the index is defined, you can go ahead and pull in the data. The index definition will have a huge impact on how the data is imported. Remember that the data is being pulled into the index as opposed to an index being applied to the data. The amount of time this takes will depend on the size, structure and location of the data, as well as the index definition. For example, indexing seven different description fields will take more time than indexing only two of them.

You may already be wondering what happens when the data in your data source changes. Azure Search has a few mechanisms for updating the index and in many cases this can be automated

FASTEST WAY TO BUILD HIGH-PERFORMING, FEATURE COMPLETE APPLICATIONS

Easy-to-use UI controls trusted by developers and enterprises worldwide

Windows, Web, and Mobile UI Controls, Reporting, and Productivity Tools

Global Xamarin Dev Days Partner

Visual Studio 2017 RC
RC Extension Sim-Ship Partners

- Extensible controls with easy-to-use universal API
- Industry's best high-performance grids, charts, and reports
- Small footprint reduces app bloat
- Global support and community

ComponentOne Studio

Download your free 30-day trial at
www.ComponentOne.com

GrapeCity

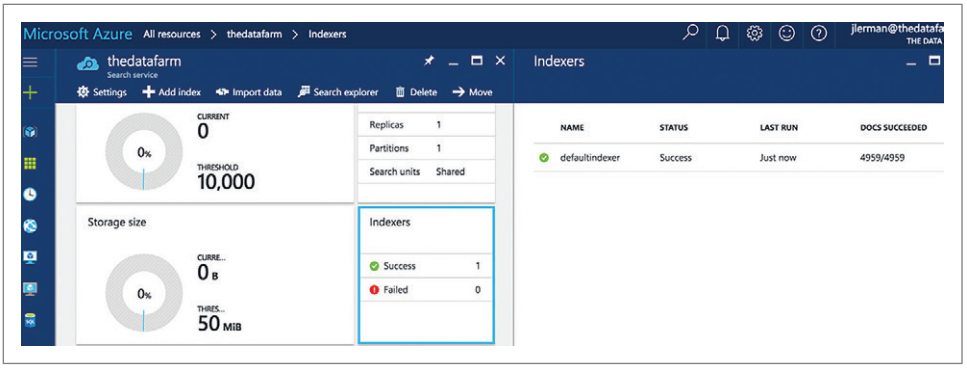


Figure 3 **The Indexer Blade in My Search Service Shows the Status of the Data Import**

as part of your Azure Search service definition. The rules and parameters around this change detection and data movement differ depending on the data source. You can learn more about this in the documentation.

Next, the wizard will ask you to create an indexer. An indexer is a crawler that reads data from the data source and populates the target index. It's what performs the initial indexing, as well as updating the index either based on a defined schedule or on demand. Because I've chosen an Azure SQL Database data source, the wizard will use a specially defined indexer that knows how to crawl an Azure SQL Database. Like every resource, the indexer needs a name. I've called mine defaultindexer, which might not be a best practice in naming, but got the job done for this demonstration.

When you click OK, the indexing will begin. The portal will pop up a notification to let you know that it's started and that you can watch its progress in the indexer blade. My import happened so quickly that by the time I looked, it had already completed. In **Figure 3**, I've scrolled down the page in the blade for thedatafarm search service. As you can see, the Indexers box is highlighted and, because I clicked that, the Indexer blade is open to the right, showing that defaultindexer finished its job, creating 4,959 search docs from the data source.

Checking Out the Searchable Documents

Now the index is ready for searching. The search explorer is a good way to get a feel for searching, as well as to test searches directly before implementing them in your code. If you've ever worked with OData, you might notice that the way you express filtering, sorting and paging uses OData syntax. Searching itself can be done using one of two query syntaxes. The default syntax is referred to as a simple syntax and the other is the Lucene query syntax.

For your first search, I recommend just clicking the Search button without putting anything into the Query string field. This returns all the data, although it will arrive in pages of 50 at a time. It's worth a look because this is an unfamiliar data source. It also gives you a feel for how the data is structured.

The results begin with a header that describes the index. The entire array of documents is wrapped in a "value" tag and each document begins with a search score value, and then each field is listed with its value.

Combining Search with OData Filter and Count

For the second search, let's stick to pure searching. Enter condominium into the query string field and hit Search. Notice the URI ends with &search=condominium.

By default, Azure Search will return paged data with 50 documents at a time, but it's pretty hard to see what you're getting here. Let's add in an OData parameter to ask for the count. Because you're now using multiple parameters, you'll need to specify that condominium is part of a search. Here's the query string:

```
$count=true&search=condominium
```

Now in the results, shown in **Figure 4**, after the index description and before the first value is listed, you can see the count of the results is 399 documents. The search looked in every field that the index defined for searching and returned a count of every document where it found the string "condominium" and then the first 50 of those 399 matching documents. If you scroll to the bottom, you'll see that there's a URI to assist you in retrieving the next 50 documents. This is a pattern you may recognize if you've worked with OData before.

To search the results pane, first make sure your cursor is somewhere in the results and then use the browser find command (for example Ctrl+F). A special search box pops up that's clearly different from the browser's search box. Type condominium and

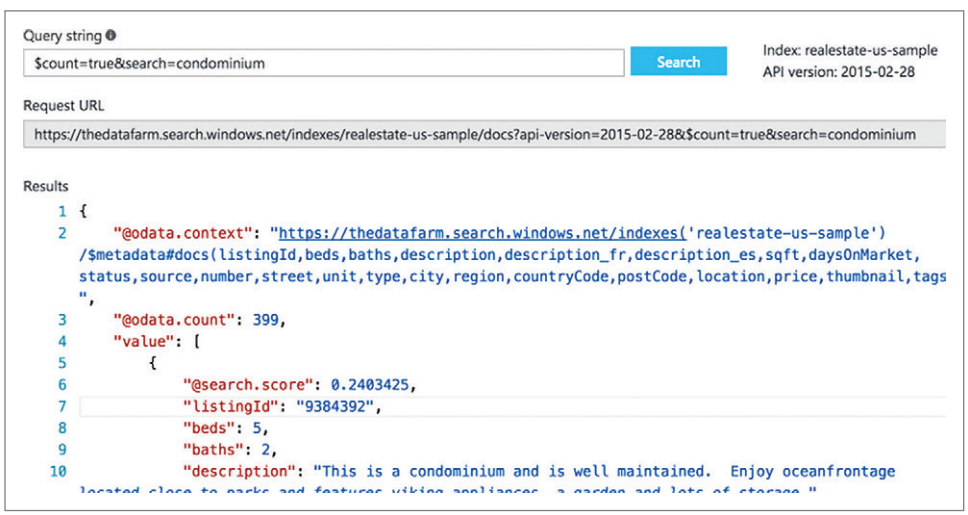


Figure 4 **The Beginning of a Result Set from a Query with OData $count in the Request; the Count Is Part of the Results Root**

click through the find results and you'll see that word pop up in the description and tag fields, and perhaps in others, as well, but I only looked at a few.

Remember that some fields were defined in the index to be filterable. Modify the query to add in a filter for three-bedroom listings; in other words, where the beds field is three. Filter is an OData parameter and requires a dollar sign in front of it. Here's the new query string using the OData syntax for filtering:

```
$count=true&search=condominium&$filter=beds eq 3
```

The count shows we're down to 73 documents.

Exploring Azure Search Even Further

At this point, I began to have a better idea of how I might want to define my index, which should be strongly influenced by what you want your app to accomplish. You may have various apps that access the same data store but require different capabilities or different data in the results. With Azure Search you can build separate indexes to satisfy those different needs, and because each index has its own set of data, the apps will not conflict with each other for resources, which could degrade performance.

I like to be able to visualize what I'm working with. Starting my Azure Search education through the portal made it possible for me to visualize the index, and visualize the results. It also allowed me to test out queries. With that experience under my belt, I have more confidence to try Azure Search in an app using the .NET Client or simply executing queries directly with REST. I can use my own databases or other sample data. One experiment I've already done was to import data from the public data set shared by the Cooper-Hewitt Museum at bit.ly/2hr0Cej. I uploaded a set of JSON documents from their objects data set into an Azure DocumentDB, then built a search service and an index from that. After this project, I discovered that one of the sample applications from the Azure team had gone in a similar direction using publicly available data from the Tate Gallery. You can play with their demo at bit.ly/2gxoHQL to witness more of the power of Azure Search, such as the use of faceted navigation. (Facets are another option to set up when you define an index.) The sample uses JavaScript and interacts with Azure Search directly through the URIs, so looking at the code on GitHub (bit.ly/2gl1ej9) isn't only educational, but also an impressive view of how simple your own logic can be because Azure Search does all of the heavy lifting for you.

I hope this first look at Azure Search will get you over any initial fear that this is a big, daunting service meant for advanced Azure experts only. Azure exists to take on the hard stuff so that you don't have to, and Azure Search is a great example of this because it takes care of the hard part of adding search capabilities into your apps. ■

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: *Pablo Castro*

Serverless Architecture with Azure Functions

Joseph Fultz and Darren Brust

From tools to machines to computers, we look for ways to automate repetitive work and standardize the context in which we work so that we can focus on high-value specialized contributions to complete tasks and solve problems. In parallel, it's clear that as the IT industry has evolved, we've strived to achieve higher density at every level of a system, from the CPU to the server farm, in hopes of attaining the highest efficiency output from our systems. A serverless architecture is the point at which those two streams converge. It's the point at which an individual's effort is most granularly focused on the specific task and the waste in the system is at a minimum.

In a serverless world, developers create solutions instead of infrastructures and monitor execution and not environment health. Finance pays for time slices, not a mostly idle virtual machine (VM) farm. In Microsoft Azure, there are many consumable services that can be chained together to form an entire solution. A new key component of this is Azure Functions, which provides the serverless compute capability in a complete solution ecosystem. In this article, we'll explore what it means to have a serverless architecture and explore Azure Functions tooling.

This article discusses:

- How serverless designs give more focus to problem solving
- Full systems that can now be created without having to manage servers
- Azure Functions that provide the serverless compute capability of a serverless architecture

Technologies discussed:

Azure Functions, Serverless Compute, Platform as a Service, WebJobs, Serverless Architecture

Serverless Architecture with Azure Functions

There are many definitions for serverless architecture. Despite the nomenclature, serverless architecture isn't code that runs without servers. In fact, servers are still very much required; you just don't have to think about them. You might think it's the next iteration of Platform as a Service (PaaS), and while close, it's not really that, either. So what, exactly, is it? Fundamentally, serverless architecture is the next evolution of cloud services, built on top of PaaS, abstracting away VMs, application frameworks, and external dependencies via bindings so that developers can focus simply on the code to implement business logic.

It's prudent to discuss the typical properties of Azure Functions and serverless architectures. Azure Functions provides the serverless compute component of a serverless architecture. As shown in **Figure 1**, Azure Functions builds on top of Azure App Service and the WebJobs SDK, adding a bit of extra magic to host and run the Azure Function code and provide some niceties such as runtime binding.

Thus, all the benefits you get by using Azure App Service are there and can be found just below the services in the settings. Additionally,

it also means that the WebJobs SDK can be used locally to create a local runtime environment. While not an architectural benefit, Azure Functions is a polyglot platform that supports a variety of languages, including Node.js, C#, PhP, Python and even Windows PowerShell. Runtimes and dependencies are handled by the platform. This is an advantage for those working in mixed environments as it lets teams with differing preferences and skills leverage the same platform for building and delivering functionality. Using Azure Functions as the serverless compute component of a serverless architecture provides several key benefits:

- **Reduced time to market:** Because the underlying infrastructure is managed by the platform, developers are free to focus on application code that implements business logic. Azure Functions might be considered a further decomposition of microservices to nanoservices. Both paradigms provide a boon to the development process as development, testing and deployment activities are narrowly focused on a small bit of functionality that's managed separately from other related, but discrete services.
- **Lower total cost of ownership:** Because there's infrastructure or OSes to maintain, developers can focus on providing value to the business. Additionally, the investment needed for DevOps and maintenance is significantly simplified and reduced.
- **Pay per execution:** A key cost savings is typically found by only paying for the consumed cycles in conjunction with increasing the functional density of the compute resources used.

As for your approach when building using Azure Functions, a key to realizing the full value is to write only the smallest unit of logic to do a single scope of work and to keep the dependencies to a minimum. When working with Azure Functions, it's best to keep to these practices:

- An Azure Function should be *single purpose* in nature. Think of them as short, concise statements rather than compound sentences.
- Operations are *idempotent* in nature. That means that the resulting state of the system from a call to an API endpoint will be unchanged if called subsequent times with the same parameters.
- Keep execution times *brief*. The goal should be to receive input, perform the desired operations and get the results to the downstream consumers. For long-running processes, you might consider Azure WebJobs or even hosting the service in Azure Service Fabric.
- In an attempt to keep the overall complexity low and execution quick, it's best to *minimize internal dependencies*. Adding too much runtime weight will slow initial load times and add complexity to the system.
- External integration through input and out *bindings*. Some common guidance given for high-performance sites is to write stateless services. This helps by not complicating or slowing a service to keep, serialize, and deserialize runtime state, as well as simplifying debugging efforts because you don't have to discover and attempt to reproduce state to figure out what happened; in stateless it's just a matter of passing the parameter values back through.

Figure 1 **Azure Functions Architecture**

Properties of serverless architecture include the following:

- The unit of work in a serverless architecture takes the form of a stateless function invoked by events.
- Scaling, capacity and infrastructure management is provided as a service.
- Execution-based billing model where you only pay for the time your code is running.

> Azure Functions provides the serverless compute component of a serverless architecture.

Some challenges of serverless architecture include the following:

- **Complexity:** While serverless architecture simplifies many things for the developer, the platform abstractions require you to rethink the way in which you build applications. Managing a monolithic application as a single unit is more straightforward than managing a fleet of purpose-built functions and the dependencies between them. This is where capabilities such as those provided by Azure API Management come into play, giving you a way to create a consistent outward facing namespace that tie together all of the discretely defined and managed functions.
- **Tooling:** Being relatively new, the tools to write, debug and test functions are still under development. As of this writing, they're currently in preview, but Visual Studio 2017 will have the tooling built-in. Also, management and monitoring tools are still evolving, but there's a basic interface for Azure Functions where you can see requests, success, errors and request-response details. There's plenty of work to do when it comes to tying in the platform with application monitoring tools and the Azure Functions team is working to grow and evolve such support.
- **Organizational Support:** It's a non-trivial consideration for some to move to a serverless paradigm. Many organizations are challenged to move to fully automated continuous

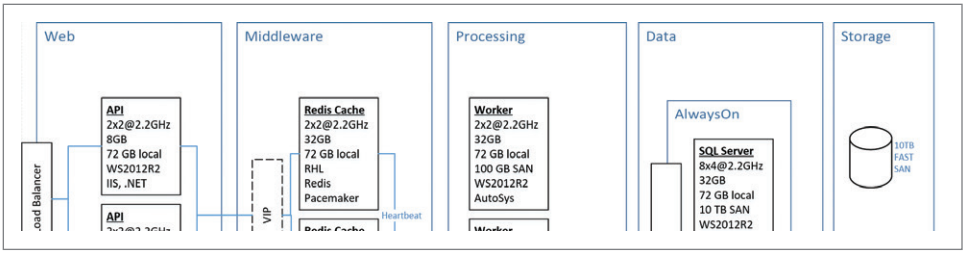


Figure 2 **Traditional Technical Architecture**

integration (CI)/continuous delivery (CD) pipeline and microservices architecture. The move to a serverless design can add to those difficulties as it'll often challenge current standards and require educating resources on what's available, how to tie it together and how to manage it.

- **No Runtime Optimization:** In a traditional design, you might optimize the execution environment to the workload, changing the type and amount for things such as RAM, swap, disk and network connectivity. With Azure Functions, only minimal changes can be made, such as which storage account to use.

Traditional Architecture vs. Serverless Architecture

A typical set of design artifacts for a system includes a logical design, technical design and software architecture. The logical design typically defines the capabilities of a system and what it does, whereas the technical design typically defines what the system is. As you move to a serverless architecture, you become more concerned about what a system does more so than what it is. In many IT shops you might find more architecture diagrams that look something like **Figure 2**.

This type of artifact is particularly important to the folks in hosting, networking, and the DBA group as they need to know what to provision and configure. However, in a Platform-as-a-Service (PaaS) environment, you're focused on the functional properties and you leave provisioning details to the platform having only to define the configuration of the PaaS services itself. The details about your input into the configuration will be kept in templates and all of the conversations you have are focused on capabilities, integration, and functionality instead of discussions about what are the optimal amounts in RAM, CPUs, and disk.

The result diagrams might be a bit closer to what you typically see in a logical diagram as is depicted in **Figure 3**. Such diagrams can start to focus on function parts and how they tie together, instead of the focus being on the configuration of application hosts. This simplification of message and clarification of purpose will help not only in technical discussions about intent and implementation, but will inevitably help convey the value of a system to a more non-technical office as the focus turns from what the system is to a discussion about what it does for someone.

Azure Functions in Action

The Azure IoT platform provides a rich set of services to support the collection and analysis of data. Our example entails an existing IoT implementation built to collect vehicle telemetry and store it in the cloud, with basic analytic functionality. We now want to explore adding new capabilities to the solution, such as being able to query the data in real time in order to find the closest vehicle to a given location. You might use this type of functionality in a free-floating car-sharing program or even to find your car in a parking lot.

We must note some caveats in this example implementation as we're purposefully taking some shortcuts in order to demonstrate the tools and platform. First, we're working directly with a single partition DocumentDB. In a more prime-time implementation we would have at minimum sharded the DocumentDB based on the expected volume of data, but we might have also chosen to do other things, such as adding Azure Redis Cache and Azure Elastic Search as the means for optimizing some of the read paths. Second, since the current DocumentDB API would require a bit more client-side processing to get to the records that we want to compare, we've taken the shortcut of just asking for the top 100 records. Partitioning and search capabilities would be a more typical path to find the records needed in a large set. In any case, you'd still use the function to find the potential records and then compare them to the given location and return the closest vehicle out of the set.

Creating the Function

At the time of this writing, the Visual Studio tooling isn't available to help speed the development process. To that end, we'll start by creating the Azure Function via the portal interface. After you start the Azure Functions creation in the Azure portal, you'll be presented with the blade to set a few settings for your function. Note the options for the App Service plan selection: Consumption Plan and App Service Plan. Choosing between these two seems like the simplest of choices, but in reality you're being presented with choosing between the old style of managing resources and the idealistic goals of a serverless architecture. In choosing App Service Plan, you must make guesses about how much processing power is needed. If you choose too much, you're paying for resources that you're not using, but choose too little and you might have issues with your implementation that ultimately impacts your customers and your bottom line. Choosing Consumption Plan is the preferred style as it leaves the scaling up and down to the system and you, as the consumer, pay for as much or as little as your app consumes.

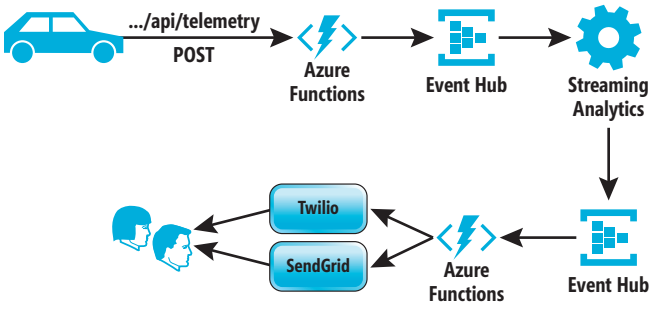


Figure 3 **Serverless Architecture**

BEST SELLER

ActiveReports 11 | from $1,567.02

GrapeCity

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

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

BEST SELLER

Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

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

BEST SELLER

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

ASPOSE
File Format APIs

Every Aspose .NET component in one package.

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

BEST SELLER

DevExpress DXperience 16.1 | from $1,439.99

DevExpress

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

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

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

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

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

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

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

GSA Schedule
Contract GS-35F-0188R

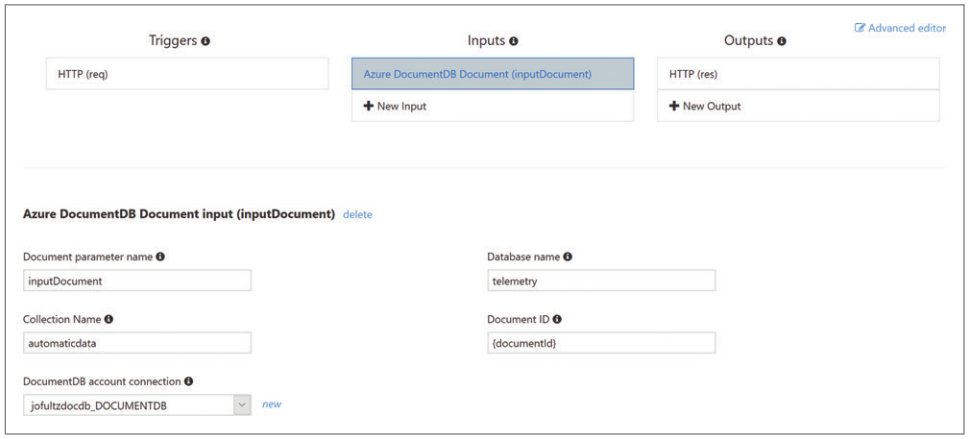


Figure 4 **Input Binding Configuration**

Your final step is to actually create the function itself. You're going to start with the WebHook premade function using C#. This will preconfigure a Trigger to prosecute the function based on receiving an HTTP request. By selecting the Integrate item on the left menu, several options can be chosen as a matter of configuration for Trigger, Input and Ouput. The Trigger selection page has some useful information on it, including information about the WebHook bindings, information about how to use the API keys, and some sample code for calling the WebHook from both C# and Node.js. Once presented with the dialog, you'll configure the Input and the Output bindings, as shown in **Figure 4**.

While you can include libraries and adapters to make calls to external systems, the system works via Bindings and has first-class support for many data sources such as EventHubs and DocumentDB. The development experience is enhanced through the Bindings infrastructure that's part of Azure Functions. Bindings abstract the actual target software infrastructure (DocumentDB, EventHubs, Storage and so on), letting the developer code to the parameter representing the target, while remaining flexible because the target can be changed by changing the configuration. Here, you've configured it to talk to the source DocumentDB. From that configuration you'll be able to write code directly against the DocumentDB client within the function and you won't have to make the connection yourself. Note the document parameter name: inputDocument. That's the variable that's passed into the function you'll use to make calls to the DocumentDB.

Here, the output options can be seen and include a number of storage, queuing and other external systems. All of the items that can be selected and configured through the UI can be accessed later via the function app Settings UI and can be configured as part of a template or through the programmatic interfaces. You're simply returning the JSON result over HTTP to the caller. Because the HTTP(res) is already set for Output with an output parameter defined, you'll accept it as is.

Developing the Code

Once Develop is selected in the left menu, you're presented with an online editor. This is good for quick iterative changes and seeing your log in real time, and it gives you an interface for triggering the WebHook Function. Tooling is underway for both Visual Studio and Visual Studio Code that will give a richer experience. If you're working with Azure Functions today, you'd want to use an IDE, and Visual Studio easily connects to the Function App through the Server Explorer.

There are a few things you need to do in editing the files. Right below the Code window there's a link for View Files. Open the File Explorer directly to the right of the Code window. First, you'll need the DocumentDB client. There are a number of libraries that are automatically available to be referenced by using the #r directive at the top. The list of those libraries, as well as a number of other pieces of developer information, can be found in the online developer reference at bit.ly/2gaWT9x. For example, you'll need access to the DocumetDB client objects because there's first-class support for DocumentDB there. You'll simply add the #r directive for that assembly at the top of the file. If a library that you need isn't included, say one you maintain and publish to NuGet, it can be added to a project.json file (package.json in Node.js).

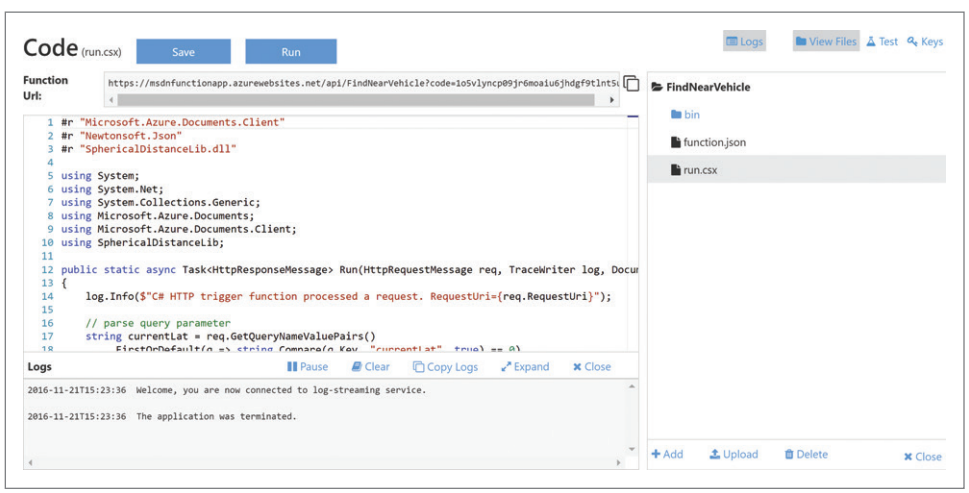


Figure 5 **Editing Function Code**

At this point you're ready to edit the run.csx file, which will hold all of your code. To do this you'll edit it directly in the online IDE for Azure Functions, as shown in **Figure 5**.

Starting with the template code, first add your own custom external library to the function, as it contains the code for the haversine function. If you have custom classes

Figure 6 **Last Known Location for Distinct Vins**

```
HashSet<string> seen = new HashSet<string>();
foreach(dynamic doc in telemDocs){
  if(seen.Add(doc.vin)){
    // Calculate the distance
    if (doc.location != null) {
      doc.distance =
        Haversine.CalculateDistance(double.Parse(currentLong),
        double.Parse(currentLat), double.Parse(
        doc.location.lon.ToString()),
        double.Parse(doc.location.lat.ToString()));
      lastKnownLocations.Add(doc);
    }
    else{
      // Location is null, so we won't take this
      // record as last known location
      seen.Remove(doc.vin);
    }
  }
}
```

or functions that aren't too large and are specific to the function, you can add them directly into the run.csx file. However, if you have reusable pieces, you'll want to go a different route and include their compiled version in a \bin folder or reference them as NuGet packages via the project.json file and reference the library with #r. Alternately, you could place the code into a different .csx file and use the #load directive.

You need a couple of functions to help determine the distance between the vehicles for which you're checking proximity and the point they passed into the function. It's been a while since school days and it turns out that you don't regularly need a haversine formula. Wikipedia provides a good reference for it at bit.ly/2gCWrgb and

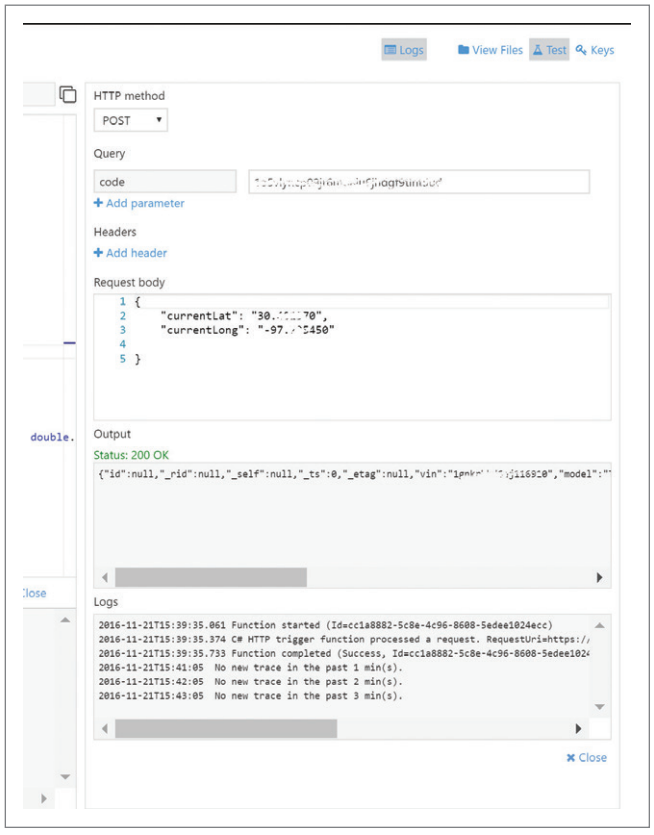


Figure 7 **Testing a Function**

we borrowed the C# function from bit.ly/2gD26mK and made a few changes. We've created the necessary functions as static members of a haversine class:

```
Namespace SphericalDistanceLib
{
  public class Haversine
  {
    public static double CalculateDistance(
      double currentLong, double currentLat,
      double vehicleLong, double vehicleLat){…}

    public static double ToRadians(double degrees){...}
  }
}
```

The code is compiled and then uploaded into a bin folder relative to the root folder of the function. In our case the path is FindNearVehicle/bin, but the bin folder must be created as it's not there by default. With that bit of setup completed, you turn your focus to the code you need. At the top of the function you'll need to ensure that you're referencing any libraries you need. In particular, you need the DocumentDB client object types, Newtonsoft, and the custom library that was uploaded to the /bin folder. These get added at the top of the file using the #r directive:

```
#r "Microsoft.Azure.Documents.Client"
#r "Newtonsoft.Json"
#r "SphericalDistanceLib.dll"
```

As previously noted, you'll grab the last 100 records based on the created_at timestamp field. DocumentDB has a nice SQL syntax that makes that pretty easy:

```
IQueryable<Document> telemDocs = inputDocument.CreateDocumentQuery<Document>(
  UriFactory.CreateDocumentCollectionUri(dbName, collectionName),
  new SqlQuerySpec("SELECT TOP 100 c.vehicle.vin, c.vehicle.model,
  c.location FROM c ORDER BY c.created_at DESC"), queryOptions);
```

You're using the Document type, which eases things a bit, because you'll cast it to a Dynamic type to add properties to the object easily. The SQL is passed in the form of a SqlQuerySpec and you'll project only the vin, model and location into your object. At this point you have to iterate the list of documents, calculate the distance using the haversine function in the external library, and determine the nearest one and return it. However, it gets a little tricky.

You need to keep track of all of the vins you've seen, because you only want the latest location record for that vin. Because you'll have ordered it in descending order, the first document is the last document you've received. You'll check the vin for null, because you're looking at vehicles that are being driven and if there's a null in the vin, you can assume that the document is invalid. If the element has a non-null value, you'll simply attempt to add the vin to a HashSet. If it's unique to the list, the addition will succeed, but if not, it'll fail and you know that you have the most recent record for that vehicle already in the list, as shown in **Figure 6**.

You add the entire document to the lastKnownLocations list, making it easy to turn around and query out the first document based on ordering by the least distance value:

```
var nearestVehicle = lastKnownLocations.OrderBy(x => ((dynamic)x).distance).First();

return currentLat == null || currentLong == null
  ? req.CreateResponse(HttpStatusCode.BadRequest,
  "Please pass a lat and lon on the query string or in the request body")
  : req.CreateResponse(HttpStatusCode.OK, nearestVehicle);
```

The first document in the ordered list can be returned, as seen in the last line, where a null param check is also handled, and the serialization of the document is handled for you.

Running the Example

The last step of this is to see it in action. At the top right of the development view there's a link with a flask icon attached to it labeled Test. Clicking this will open the testing UI, as shown in **Figure 7**. Within it you can change the HTTP method, add parameters, add headers and set the body to be passed to the selected HTTP method. We typically use Postman when we're doing a lot of this work, because we have a library of tests. However, the built-in testing facilities for HTTP functions are excellent and immediately available for quick iterative work.

We've grabbed the latitude and longitude and formatted it into the expected JSON format in the Run window. Once Run is clicked, any output from calls to the log object and general log output can be seen in the Logs window with the output and the status seen on the bottom right in the Output window. Taking note of the timing on the log output it looks like it took around 250 ms for our function to run. That's well within the execution model for which we're striving with Azure Functions: singlepurpose and relatively short execution times. Pulling the content out of the Output window and formatting it, we can see a lot more clearly that we have the vehicle, the timestamp when the location was recorded, the location and the distance:

```
{
  "vin": "wmwrxxxxxxxx54251",
  "model": "Cooper Hardtop",
  "location": {
    "lat": 30.4xxxxxx,
    "lon": -97.8xxxxxx,
    "accuracy_m": 22.505,
    "ts": 1473116792970
  },
  "distance": 13.552438042837085
}
```

When we did the distance calculation we gave the circumference of the Earth in kilometers, so the distance that's represented in the return is about 13.6 km.

> Azure Functions turns the focus to the value of the code and away from managing infrastructure.

Wrapping Up

In the example, we used a mixture of online tools to develop and prosecute Azure Functions, but a more realistic approach for a development team would be to develop locally and set up continuous integration and deployment via a Git repo. Using the WebJobs.Script and WebJobs SDK, you can set up the ability to develop and run Azure Functions locally. A good walk-through of how to do that can be found at bit.ly/2hhUCt2. You'll also find there are a number of different sources that can be configured as the source for deployments.

Azure Functions is the new kid on the block in the Azure platform. It's a key ingredient of serverless compute that's needed to achieve the benefits of a cloud PaaS implementation. Azure Functions turns the focus to the value of the code and away from managing infrastructure. The platform, tools support, and language support continues to evolve, but it already supports a number of languages and can be tied into your CI/CD pipeline. More information can be found at bit.ly/2h7lo4C. If you're not already working with Azure Functions, you should give it a try. You can get started at bit.ly/2glBJnC. ■

The Shift to Serverless Compute

Serverless compute represents a fundamental paradigm shift in how we think about cloud computing and building applications for the cloud. Serverless computing provides developers with a completely abstracted and infinitely scalable environment to execute their code and provides a payment model that is purely focused on code execution.

Serverless compute can be looked at as the next step in the evolution of Platform as a Service (PaaS). It fulfils the PaaS promise of abstracting the application infrastructure from the code and providing auto-scale capabilities. The major differentiator for serverless is the per-execution pricing model (rather than paying for the time the code is hosted) and instant, unlimited scale. Serverless compute provides a *fully managed* compute experience (zero administrative tasks), *instant unlimited* scale (zero scale configurations) and *reacts to events* (real-time processing). This enables developers to develop a new class of applications that scale by design and, ultimately, are more resilient.

Developers building serverless applications use serverless compute like Azure Functions, but also use an ever growing number of fully managed Azure or third-party services to compose working end-to-end serverless solutions. Developers today can easily set up and run complete serverless architectures that scale with ease, by design and at little cost. Developers also are free from the burden of managing and monitoring infrastructure; they can focus on their business logic and solve problems related to the business and not to the maintenance of the infrastructure running their code.

Serverless is here to change the way we think about building applications and is here to stay for a long time. Join the conversation at bit.ly/2hhP410 and at bit.ly/2gcbPzr. Submit feature requests to bit.ly/2hhY3jq and reach out to us on Twitter: @Azure with the hashtag #AzureFunctions. —*Yochay Kiriaty*

Darren Brust *is a cloud solution architect at Microsoft where he spends most of his time assisting enterprise customers as they transition to Microsoft Azure. In his free time, you're likely to find him at one of his three children's sporting events or a local Austin coffee shop consuming copious amounts of coffee. You can reach him at dbrust@microsoft.com or on Twitter: @DarrenBrust.*

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

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

Inside the Azure App Service Architecture

Yochay Kiriaty and Stefan Schackow

Azure App Service is considered an excellent Platform as a Service (PaaS), offering an application platform for developers to build Web, mobile and API applications. Its offerings range from simple marketing and digital presence applications to scalable e-commerce solutions and hyper-scale, customizable applications.

App Service is fully managed, which means no administrative tasks are required to manage underlining compute infrastructures (servers) on which your applications run. You don't need to worry about the underlining server maintenance as the platform patches the OS and frameworks for you. Your application runs on virtualized servers, but you should only care for setting the maximum number of server instances on which you want your application to run. The platform handles scaling when your application needs additional compute resources and, at the same time, it handles the load balance traffic across multiple instances of your application.

While the App Service team is doing its best to hide any underlining implementation details, it's good to be aware of how things work under the cover. This article covers the basic internal architecture of App Service (how the service is built and operates) and offers a few best practices for certain scenarios.

This article discusses:

- Azure App Service scale units, the logical "building blocks" of the service
- What is an App Service Plan and best practices for scaling App Service Plans
- Using per-app scaling to increase application hosting density
- Best practices for optimizing usage of outbound network connections

Technologies discussed:

Azure App Service, App Service Plans, Per-Application Scaling, Deployment Slots, Outbound Network Port Usage

Global and Geo-Distributed Architecture

Cloud computing quickly scales and has endless capacity. Cloud scale can be explained as looking on a computer screen. Look from afar and you see a clear and smooth picture; when you take a closer look, you notice the image on the screen is comprised of many little pixels. The cloud, like an image, is comprised of many servers. App Service clusters bunches of servers into a single unit called a "scale unit" (or a "stamp"). There are many such scale units across the globe in Azure datacenters.

As part of Azure, App Service has a global footprint. In every Azure-supported region, you'll find App Service scale units running customers' workloads (applications) and sets of regional control units. Control units are transparent to the customer (until they malfunction) and considered part of the platform. There's one special control unit that's being used as a gateway for all management API calls. For example, when a customer makes a request to create a new application, either through the portal, command-line interface or directly through the Azure REST API, the request is routed to a central Azure endpoint (management.azure.com). The Azure Resource

Manager, or ARM (bit.ly/2i6UD07), lets you work with different Azure resources in your application as a single group. The API defined by ARM lets you manage Azure resources. ARM doesn't actually manage individual resources. Each service in Azure has its own management API implementation that is proxied by ARM. In the case of App Service, ARM forwards any App Service API calls to App Service Geo-Master.

The Geo-Master has context about all scale units worldwide. For example, when you create a new App Service application (or Web site), Geo-Master finds the most suitable scale unit for your application and then forwards the create request to the appropriate scale unit. The scale unit is now tasked with provisioning a new app and allocating any required resources. **Figure 1** shows the flow of creating a new app.

Here's the process to create a new app:

1. User makes a request to create a new site.
2. ARM makes sure user has access to the resource to allow the given operation (create in this case) and forwards the requests to App Service Geo-Master.
3. Geo-Master finds the best suitable scale unit for the user's request and forwards the request.
4. The scale unit creates the new application.
5. Geo-Master reports success on the create request.

While it's important to understand that App Service has many scale units, your application usually runs within a single App Service scale unit. Even when using Azure Traffic Manager to run in multiple regions, your application runs in two or more separate scale units. However, from the individual scale unit point of view, your app is constrained to a single scale unit.

What Is an App Service Scale Unit?

An App Service scale unit is a collection of servers that host and run your applications. A typical scale unit can have more than 1,000 servers. The clustering of servers enables economy of scale and reuse of shared infrastructure. The fundamental building block of App Service scale unit is an Azure Cloud Service deployment (remember, App Service first released as a preview in June 2012). Any given scale unit is autonomous and can operate on its own.

Scale Unit Main Building Blocks

The main functionality of scale unit is to host and run customer applications. Applications run on Windows servers and are referred to as Web Workers or Workers for short. The majority of servers in a given scale unit are Workers. However, a unit of scale includes several additional support servers required to achieve the functionality provided by App Service. Support servers have roles and each role is deployed on multiple instances for redundancy and scale.

Front End

The front end is a layer seven-load balancer, acting as a proxy, distributing incoming HTTP requests between different applications and their respective Workers. Currently, the App Service load-balancing algorithm is a simple round robin between a set of servers allocated for a given application.

Web Workers

Workers are the backbone of the App Service scale unit. They run your applications.

With App Service, you can choose how you want to run your application. You can select running your application on shared or dedicated servers. You do so by selecting an App Service Plan. An App Service Plan defines a set of capabilities, features and server allocations. A Shared Worker hosts applications from multiple different customers where Dedicated Workers are guaranteed to run one or more applications of a single customer. There are several dedicated server types and sizes from which you can choose. The larger the server size, the more CPU and memory resources are available for allocated applications. The App Service Plan defines the amount of pre-allocated servers for your application.

An App Service scale unit has several pools of Workers pre-provisioned and ready to host your applications, as illustrated in **Figure 2, Section 1**. When you define your dedicated App Service Plan to a size of two servers, App Service allocates two servers, as illustrated in **Figure 2, Section 2**. Next, when you scale out your App Service Plan—for example, adding two more Workers—available Workers are allocated from a pool of ready-to-go Workers, as illustrated in **Figure 2, Section 3**. Because Workers are already pre-provisioned and warm, all that needs to happen is for your application to get deployed on the Worker. Once the app is deployed, the Worker is inserted into the rotation and the front end can allocate traffic to it. This whole process usually takes a few seconds.

Figure 2, Section 4 shows multiple App Service Plans, identified as multi-colored rectangles, each representing an App Service Plan that can belong to multiple customers.

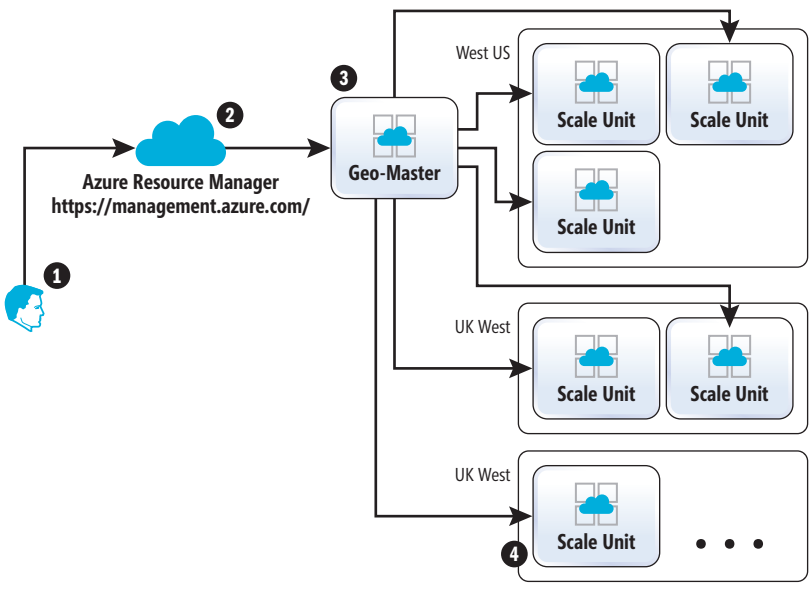


Figure 1 **Global Distribution of App Service Scale Units**

File Servers

Any app needs storage to hold content such as HTML, .js files, images, or code files, and any other content required for the application to work. A file server mounts Azure Storage blobs and exposes them as network drives to the Worker. A Worker, in return, maps such network drives as local, allowing any application running on any given Worker to use the "local" drive, just like you would expect if the application were running on a server using its local disk. Any file-related read/write operation performed by the application passes through a file server.

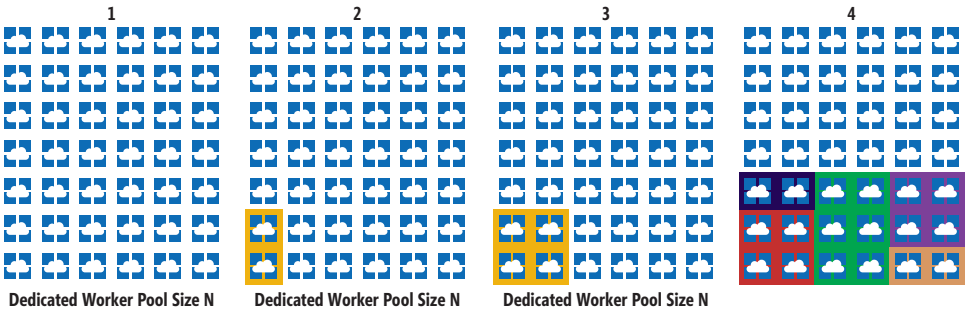


Figure 2 **Server Application Process in App Service Scale Unit**

API Controllers

API controllers can be viewed as an extension to App Service Geo-Master. While the Geo-Master is aware of all App Service applications across all scale units, it's the API controller that actually performs the management operation that affects your applications. Geo-Master delegates API fulfilment to a given scale unit via the API controllers. For example, when Geo-Master passes an API call to create a new application, the API controller orchestrates the required steps to create the application at the scale unit. When you use the Azure portal to reset your application, it's the API controller that notifies all Web Workers currently allocated to your application to restart your app.

Publishers

Azure App Service supports FTP access to any application. Because app content is stored in Azure Storage blobs and mapped by file servers, App Service has a Publisher role to expose FTP functionality. The Publisher role lets customers use FTP to access their application content and logs.

It's important to note that there are many other ways to deploy applications other than FTP. One of the common deployment methods is Web Deploy (from Visual Studio) or any of the supported Continues Deployment options such as Visual Studio Release Manager or GitHub.

SQL Azure

Each App Service scale unit uses Azure SQL Database to persist application metadata. Each application that's assigned to a given unit of scale has a representation in a SQL Database. The SQL Database is also used to hold runtime information about applications.

Data Role

All roles require data found in the database to operate. As examples: a Web Worker needs site configuration information when launching an app; front ends need to know which servers are assigned to run a specific application in order to correctly forward HTTP requests to the appropriate servers; and controllers read and update data from the database based on API calls made by customers. The data role can be described as a cache layer between SQL Database and all other roles in a given unit of scale. It abstracts the data layer (SQL Database) from the rest of the roles, improving scale and performance, as well as simplifying software development and maintenance.

Less-Than-Obvious Best Practices

Now that you know how Azure App Service is built, we'll review several tips and tricks from the App Service team. These are hands-on lessons learned by App Service engineering teams from numerus customer engagements.

Controlling Density

A majority of customers run a low number (less than 10) of applications per App Service Plan. However, there are many scenarios where customers are running many more applications. It's important to prevent accidentally over saturating the compute capacity of the underlying servers.

Let's start with the basic hierarchy of applications and their compute resources. There are two Web apps and one mobile back-end app associated with this App Service Plan. The plan is set to two servers.

By default, all applications contained in a given App Service Plan run on *all* the available compute resources (servers) allocated to that Service Plan. All three applications run on both servers. In the simple case where an App Service Plan has a single server, it's simple to understand: All of the applications in the App Service Plan run on a single server.

It's somewhat less intuitive as to what happens when there are multiple compute resources allocated to your App Service Plan. As an example, if a single App Service Plan has 10 compute resources, then every app in the application service will run on *every* compute resource. If there are 50 applications in the App Service Plan, all 50 will be running on the first server, and the same 50 will be running on the second server, and so on, to the 10th server, which will also be running all 50 apps.

For some scenarios, where your application requires a lot of compute resources, usually to handle an increase in incoming HTTP requests, you'll want to have your application running on all available servers. However, sometimes this is an unintended consequence that occurs when an App Service Plan is scaled out from one server to many. If an App Service Plan is under CPU and/or memory pressure due to a large number of applications running in it, increasing the number of servers in that App Service Plan won't address the problem.

Instead, consider the traffic distribution for each application and separate the long tail of low-volume apps into a separate App Service Plan. Consider running high-volume apps in separate App Service Plans. Using the 50 app example earlier, after analyzing traffic patterns you might end up with the following allocation to App Service Plans and compute resources:

- 40 low-volume applications remain in a single App Service Plan running on one compute resource.
- Five mid-to-low volume applications use a second App Service Plan running on one compute resource.
- The remaining five applications are found to have high-volume usage. Each application is deployed onto a separate App Service Plan. An auto-scale rule is set on each App Service Plan to use a minimum of one compute resource, with rules to scale in/out based on CPU and memory utilization.

The net result of this approach is that the 50 apps use seven compute resources at a minimum, with the five high-volume applications each having the necessary headroom to independently scale out on-demand based on load.

Per-App Scaling

Another alternative for running large numbers of applications in a more efficient manner is to use the per-app scaling feature of Azure App Service. The document at bit.ly/2iQUm1S covers per-app scaling in detail. Per-App scaling lets you control the maximum number of servers allocated to a given application, and you can do so per application. In this case, an application will run on the defined maximum number of servers and not on all available servers.

Using the earlier 50 app example, with per-app scaling enabled for the App Service Plan, all 50 apps can be assigned to the same App Service Plan. Then, the scaling characteristics of individual apps can be modified:

- 40 low-volume applications set to run on a maximum of a single server each.
- Five mid- to low-volume applications set to run on a maximum of two servers each.
- Five remaining high-volume applications set to run on a maximum of 10 servers.

The underlying App Service Plan can start out with a minimum of five servers. And then auto-scale rules can be set to scale out as needed based on memory pressure vs. CPU.

Azure App Service will automatically handle assignment of applications to compute resources. The service will also automatically handle constraining the maximum number of running application instances based on the *number of Workers* setting for each individual application. As a result, increasing the numbers of Workers in the App Service Plan will not result in 50 app instances spinning up on each new available virtual machine.

To summarize, per-app scaling "packs" applications onto the underlying servers associated with an App Service Plan. Per-app scaling *does not* result in every application running on every compute resource associated with an App Service Plan as described earlier.

Scale Unit Components

It might seem as if scale unit components are highly dependent on each other. However, by design, they're loosely coupled. An application that's currently running (actively serving HTTP traffic) on a given Web Worker can continue to serve HTTP traffic even if other roles in the scale unit are malfunctioning.

For example, a publisher not working properly might hinder FTP access, but that doesn't affect an application's HTTP traffic or other deployment options. If the controller has a bug preventing creation of new applications, it doesn't mean apps already assigned to the scale unit stop working.

Application Slots

App Service has a feature called "deployment slots" (bit.ly/2iJzv3f). In a nutshell, a deployment slot enables you to have another application (slot) other than your production app. It's another application that you can use to test new code prior to swapping into production.

Application slots is among the most used feature in App Service. However, it's important to understand that each application slot is also an application in its own right. This means application slots can have custom domains associated with them, different SSL certificates, different application settings and so on. It also means the assignment of an application slot to an App Service Plan can be managed separately from the App Service Plan associated with the main production slot.

By default, each application slot is created in the same App Service Plan as the production slot. For low-volume applications, and/or applications with low CPU/memory utilization, this approach is fine.

However, because all applications in an App Service Plan run on the same servers, this means by default all of an Application's Slots are running on the exact same underlying server as production. This can lead to problems such as CPU or memory constraints if you decide to run stress tests against non-production slots, which run on the same server as your production application slot.

If resource competition is scoped just to scenarios such as running load tests, then temporarily moving a slot to a different App Service Plan, with its own set of servers, will do the following:

- Create additional App Service Plan for the non-production slots. Important note: Each App Service Plan needs to be in the same resource group and same region as the production slot's App Service Plan.
- Move a non-production slot to a different App Service Plan and, thus, a separate pool of compute resources.
- Carry out resource-intensive (or risky) tasks while running in the separate App Service Plan. For example, load tests can be run against a non-production slot without negatively impacting the production slot because there won't be any resource contention.
- When the non-production slot is ready to be swapped into production, move it back to the same App Service Plan running the production slot. Then the slot swap operation can be carried out.

Deploying to Production with No Downtime

You have a successful application running on an App Service Plan and you have a great team to make updates to your application on a daily basis. In this case, you don't want to deploy bits directly into

GROUPDOCS
Document Manipulation APIs

Manipulating Files?

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

GroupDocs.Total

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

Try for Free

.NET Libraries

Java Libraries

Cloud APIs

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

Visit us at www.groupdocs.com

GROUPDOCS
Document Manipulation APIs

production. You want to control the deployment and minimize downtime. For that you can use your application slots. Set your deployment to the "pre-production" slot, which can be configured with production setting, and deploy your latest code. You can now safely test your app. Once you're satisfied, you can swap the new bits into production. The swap operation doesn't restart your application and in return the Controller notifies the front-end load balancer to redirect traffic to the latest slots.

Some applications need to warm up before they can safely handle production load—for example, if your application needs to load data into cache, or for a .NET application to allow the .NET runtime to JIT your assemblies. In this case, you'll also want to use application slots to warm up your application prior to swapping it into production.

We often see customers having a pre-production slot that's used to both test and warm up the application. You can use Continuous Deployment tools such as Visual Studio Release Manager to set up a pipeline for your code to get deployed into pre-production slots, run test for verification and warm all required paths in your app prior to swapping it into production.

Scale Unit Network Configuration

The App Service scale unit is deployed on Cloud Service. As such, it implies some network configuration and functionality that you might become familiar with to fully understand any network effects on your apps.

The scale unit has a single Virtual IP (VIP) address exposed to the world. All applications allocated to a given unit of scale are serviced through this VIP. The VIP is the representation of the Cloud Service on which the App Service scale unit is deployed.

App Service applications only serve HTTP (port 80) and HTTPS (port 443) traffic. Each App Service application has default built-in HTTPS support for the azurewebsites.net domain name. App Service supports both Server Name Indication (SNI) and IP-based Secure Sockets Layer (SSL) certificates. In the case of IP-based SSL, a given application is allocated a dedicated IP address for only inbound traffic, which is associated with the Cloud Service deployment. Please note: Front ends terminate SSL connection for all HTTPS requests for all applications and any type of certificate. The front end then forwards the request to the designated worker for a given application.

Public VIP

By default, there's a *single* public VIP for all inbound HTTP traffic. Any app is addressable to a single VIP. If you have an app on App Service, try running nslookup command (from Windows or PowerShell console) and see the result. Here's an example:

```
#1 PS C:\> nslookup awesomewebapp.azurewebsites.net
#2 Server:  UnKnown
#3 Address:  10.221.0.3
#4 Non-authoritative answer:
#5 Name:    waws-prod-bay-001.cloudapp.net
#6 Address:  168.62.20.37
#7 Aliases:  awesomewebapp.azurewebsites.net
```

Here's a review of the output of awesomewebapp.azurewebsites.net:

- Line #1 runs an nslookup querying resolution for awseomwebapp.azurewebsites.net.
- Line #5 shows the domain name of the scale unit running awseomwebapp app. You'll notice that an App Service scale unit is deployed on Azure Cloud Service (by the cloudapp.net suffix). WAWS stands for Windows Azure (when Azure was still called Windows) Web sites (the original name of App Service).
- Line #6 shows the VIP of the scale unit. All the applications that are hosted and running on waws-prod-bay-001 (Line #5) are addressable on the public VIP designated.
- Line #7 shows all domain aliases mapped to the same IP address.

Outbound VIPs

Most likely your application is connected to other Azure and non-Azure services. As such, your application makes outbound network calls to endpoints not on the scale unit of your application. This includes calling out to Azure services such as SQL Database and Azure Storage. There are up to five VIPs (the one public VIP and four outbound dedicated VIPs) used for outbound communication. You can't choose which VIP your app uses, and all outbound calls from all apps in scale unit are using the five allocated VIPs. If your application uses a service that requires you to whitelist IPs that are allowed to make API calls into such a service, you'll need to register all five VIPs of the scale unit. To view which IPs are allocated to outbound VIPs for a given unit of scale (or for your app from your perspective) go to the Azure portal, as shown in **Figure 3**.

If you're looking for a dedicated set of inbound and outbound IPs, you can explore this by using a fully isolated and dedicated App Service Environment at bit.ly/2hVRSIR.

IP and SNI SSL

App Service supports IP-based SSL certificates. When using IP-SSL, App Service allocates to your application a dedicated IP address for only in-bound HTTP traffic.

Unlike the rest of Azure dedicated IP addresses, the IP address with App Service via IP-SSL is allocated as long as you opt to use it. You don't own the IP address and when you delete your IP-SSL, you might lose the IP address (as it might be allocated to a different application).

App Service also supports SNI SSL, which doesn't require a dedicated IP address and is supported by modern browsers.

Network Port Capacity for Outbound Network Calls

A common requirement for applications is the ability to make outbound network calls to other network endpoints. This includes calling out to Azure internal services such as SQL Database and Azure Storage. It also includes cases where applications make calls to HTTP/HTTPS API endpoints—for example, calling a Bing Search API or calling an API "application" that implements back-end business logic for a Web application.

In almost all of these cases, the calling app running on Azure App Service is implicitly opening a network socket and making outbound calls to endpoints that are considered "remote" from an Azure Networking perspective. This is an important point because calls made *from* an app running on Azure App Service *to* a remote

endpoint rely on Azure Networking to set up and manage a table of Network Address Translation (NAT) mappings.

Creating new entries in this NAT mapping takes time and there's ultimately a finite limit on the total number of NAT mappings that can be established for a single Azure App Service scale unit. Because of this, App Service enforces limits on the number of outbound connections that can be outstanding at any given point in time.

The maximum connection limits are the following:

- 1,920 connections per B1/S1/P1 instance
- 3,968 connections per B2/S2/P2 instance
- 8,064 connections per B3/S3/P3 instance
- 64K max upper limit per App Service Environment

Applications that "leak" connections invariably run into these connection limits. Applications will start intermittently failing because calls to remote endpoints fail, with the failures sometimes correlating closely to periods of higher application load. You'll frequently see errors like the following: "An attempt was made to access a socket in a way forbidden by its access permissions aaa.bbb.ccc.ddd."

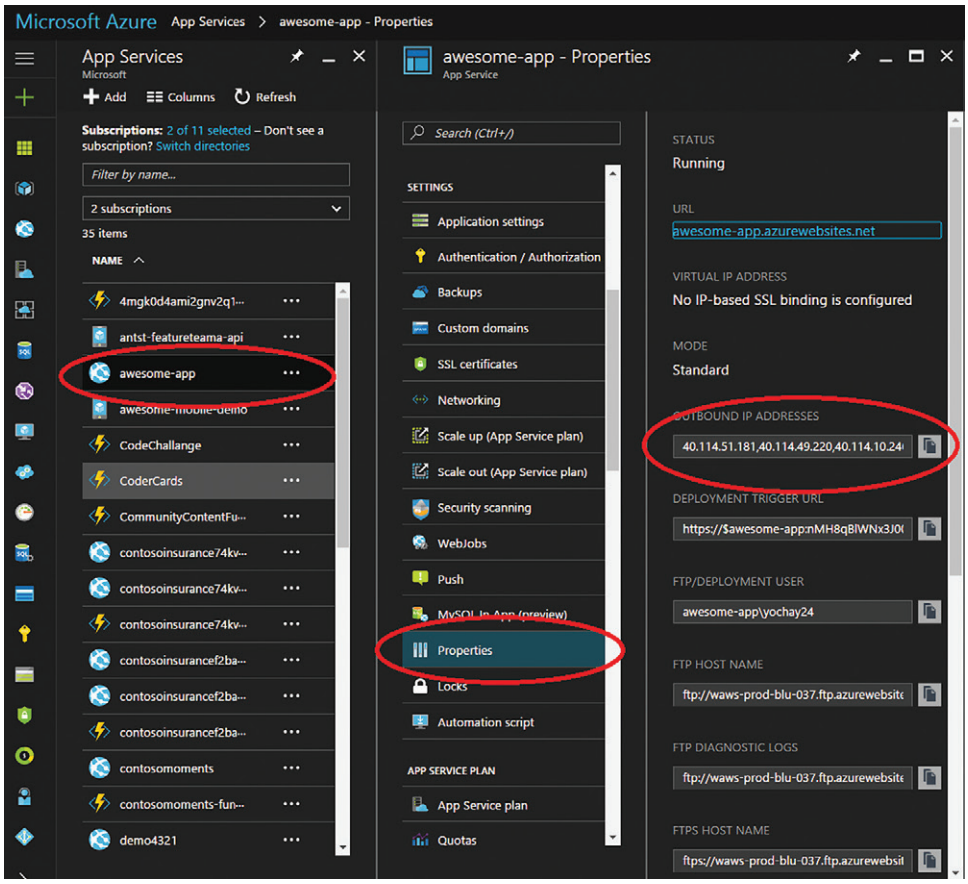


Figure 3 **App Service Application Outbound IP Address View in Azure Portal**

The likelihood of running into this problem can be substantially mitigated with a few best practices:

- For .NET applications using ADO.NET/EF, use database connection pooling.
- For php/mySql, use persistent database connections.
- For Node.js applications making outbound HTTP/HTTPS calls, configure keep-alives so that outbound connections are reused. This configuration is described at bit.ly/2iGrcoo.
- For .NET applications making outbound HTTP/HTTPS calls, pool and reuse instances of System.Net.Http.HttpClient or use Keep-alive connections with System.Net.HttpWebRequest. Note: Remember to increase the System.Net.ServicePointManager.DefaultConnectionLimit because you'll otherwise be limited to two concurrent outbound connections to the same endpoint.

There are additional limitations imposed by the App Service Sandbox. These are lower-level limits and constraints for Azure App Service and you can read about them in more detail at bit.ly/2hXJ6IL.

Wrapping Up

Azure App Service provides a rich PaaS offering for Web, mobile and API applications. While the service has a lot of moving internal parts, these are abstracted away so that developers can concentrate on writing great apps while the service handles the complexities of running those apps at global scale.

Many of the best practices for App Service revolve around application scaling. Having a good understanding of how applications map to Web Workers in an App Service Plan is important as you increase the number of your apps running on the service, while still maintaining an optimal scaling configuration.

Given the cloud-first world we operate in, Azure and Azure App Service are always constantly evolving at a rapid pace. Many new innovations are expected in 2017. ■

Yochay Kiriaty *is a principal program manager on the Microsoft Azure team, specifically driving Web, mobile, API and functions experiences as part of the Azure App Service Platform. Kiriaty has been working with Web technologies since the late 90s and has a passion for scale and performance. You can contact him at yochay@microsoft.com and follow him on Twitter: @yochayk.*

Stefan Schackow *is a program manager on the Azure App Services team who has worked on the Web app cloud offering since its earliest days. In Azure, he leads a team of program managers who work on the development and deployment of Azure App Service, as well as the development of Microsoft's on-premises/cloud hybrid products (Azure Pack and Azure Stack). He can be contacted at stefsch@microsoft.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *Eduardo Laureano and Nir Mashkowski*

Driving Development with Active Monitoring of Apps and Services

Kraig Brockschmidt, Thomas Dohmke and Alan Cameron Wills

A perfect world for writing a mobile app would be one in which, first, you know exactly what your customers want the very instant those desires arise in their minds and, second, the code you write to fulfill those needs is delivered instantly to those customers. In short, there's no gap whatsoever between customer needs, development and delivery.

Complete fantasy, right? If you've been around software development for, well, at least an hour, you know that reality comes nowhere close to this scenario. And yet every developer has probably encountered such a seamless loop, because it's exactly what you experience when you write software for yourself. In fact, it's probably what got you hooked on the magic of coding in the first place. You thought up an idea, wrote some code, pressed F5 and voila! Something amazing happened that you could control and shape and play with to your utmost heart's desire … often very late into the night.

This article discusses:

- Why monitoring is essential for DevOps
- Monitoring as a culture, not just the use of tools
- Monitoring mobile apps with HockeyApp
- Monitoring Web services and applications with Application Insights

Technologies discussed:

HockeyApp, Application Insights, Visual Studio Team Services

What probably happened next is that you started writing software for *other* people, which introduced various time lags into the process. Compared with pressing F5, the time it takes to get code changes delivered to external customers is a veritable eternity, hence source control, continuous integration, and continuous delivery to optimize the process, as discussed in the previous articles in this series. The same can be said about communication—compared with writing software for yourself, when you're both customer and developer, the challenge of clearly understanding what customers need and what they experience is almost like traveling to a distant galaxy!

Thus, although you've done your best to deliver apps and services that meet customer needs, and tried to prevent defects from reaching those customers, the truth is that some defects will still make it through and some customer needs will remain unfulfilled. This is where *monitoring*, the last stage of the release pipeline illustrated in **Figure 1**, becomes the primary concern. Here, the focus shifts from delivery to *listening*, which means being aware of bugs and issues that affect customers, and to *learning*, which means discovering unfulfilled needs and validating all the assumptions that went into the creation of the release in the first place. What you hear and what you learn feeds back around to the beginning of the release pipeline to drive the next series of code changes.

In short, monitoring closes the loop between releases, ensuring a continuous stream of value being delivered to customers, and that it's the right value. As such, it's an essential practice in the continuous validation of performance, which is what DevOps is all about.

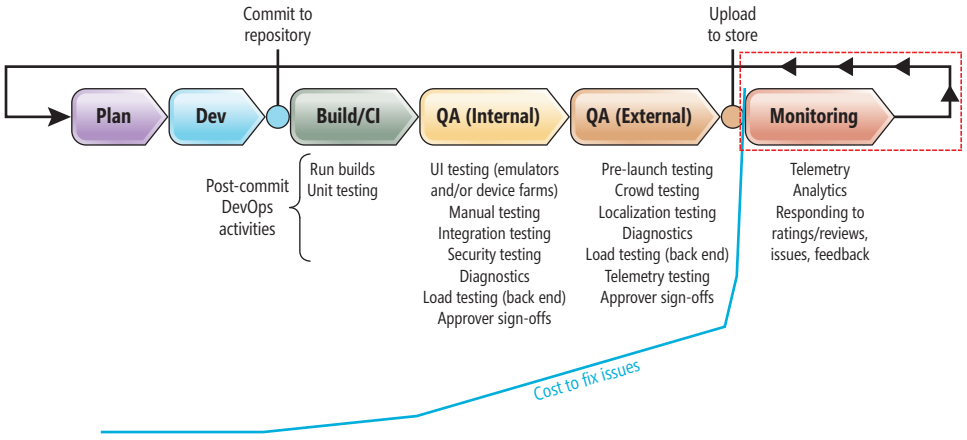


Figure 1 **Monitoring is Essential to Understand What Should Drive Subsequent Releases**

With the Microsoft DevOps stack for mobile, HockeyApp delivers monitoring for mobile apps, and Application Insights provides monitoring for Web applications and services. As we'll see, both provide multiple ways to listen to what's happening to the software out in the wild.

The Monitoring Culture

In Kraig's last article (msdn.com/magazine/mt791797), he spoke about continuous deployment as a culture, because achieving that level of delivery optimization requires a deep commitment to code reviews and automated testing. It also means a commitment to monitoring:

- Because no suite of tests is perfect, some defects *will* get through to production, so your team must actively monitor crash reports, telemetry, load imbalances, ratings, reviews, and direct customer feedback. This includes monitoring the app's performance on each supported platform, and monitoring the health of your back-end services.
- Your team must be committed to triaging and prioritizing issues and feeding them into the dev backlog so that corrections quickly get into subsequent releases.

Furthermore, monitoring is as much a matter for your test and staging environments as it is for production. Code you write to monitor the app and collect telemetry is just as prone to bugs as any other code, and requires time to debug and troubleshoot before release. Monitoring in test and staging environments also gives your team the chance to test your processes for handling crashes and feedback.

For example, custom telemetry that you collect from apps and services is typically designed to answer questions about customer usage and whether you're meeting certain business goals (see Kraig's post, "Instrumenting Your App for Telemetry and Analytics," at bit.ly/2dZH5Vx and his Build 2014 presentation, "Understanding Customer Patterns," at bit.ly/2d1p6fC). Defects in your telemetry code, however, don't affect customers. Instead, telemetry defects can easily invalidate all the data you collect. High-quality data is everything, so it's essential to fully exercise telemetry collection in both test and staging environments, keeping that data separate from production data, of course, and to scrutinize it for any collection and tabulation errors.

Test and staging environments also help you test feedback mechanisms in the app. When using HockeyApp for distribution to pre-launch testers, encourage those testers to provide feedback through the app itself, because that's how your production customers will be doing it.

Most important, though, is that monitoring your test and staging environments is how you practice being responsive to crashes, customer feedback, and issues revealed by telemetry. Put simply, monitoring is both listening and responding, because responsiveness proves you really are listening. Responsiveness, too, means acknowledging feedback and then acting on it by setting up a regular process to get appropriately prioritized issues into your backlog so your developers can get working on them.

Establishing this process isn't just an afterthought—you should start well before your first release, right alongside setting up build servers and release pipelines. In doing so, you can practice and hone your process with your test and staging environments preparation for your first release, because once you start making releases, monitoring must be continual.

> Monitoring your test and staging environments is how you practice being responsive to crashes, customer feedback, and issues revealed by telemetry.

A strong monitoring process can even, to some extent, make up for initial weaknesses in other forms of testing. It takes time and expertise to create really good automated tests, which could be challenging within the constraints of your release schedule or the availability of qualified candidates. Monitoring, on the other hand, requires less-specialized skills, and setting your team up for monitoring activities is generally quicker and simpler than writing a full suite of tests. Oftentimes, too, your early adopters are likely to be more forgiving about defects if they trust that you're listening and working hard to address them. Good monitoring, then, gives you the opportunity to get your apps and services out to customers quickly, and to start collecting valuable telemetry and feedback, before necessarily having fleshed out your testing processes.

Monitoring with HockeyApp

The particular challenge of mobile apps (and desktop apps, too) is that they're installed on millions of devices and device configurations

that are out of your control. In contrast to a Web service, you can't simply sign into the user's mobile device and analyze a log file or attach a debugger. As a result, you need another system that can retrieve crash and usage information from those devices (respecting privacy, of course), and upload that data to a central service for analysis. That's what HockeyApp is for.

In addition to providing valuable services for pre-launch distribution (as discussed last time), HockeyApp also helps you continuously monitor each release of your app throughout its lifecycle. You can collect sessions, events, crashes, and feedback starting with your first prototype, and then continue collecting information in your beta versions, release candidates, and finally the live app. This continuity gives you valuable data at every step of the process, and by monitoring from the earliest stages you'll be able to catch defects that made it through other pre-release testing processes. Thus, you'll be aware of defects as soon as they affect live customers, which helps you prioritize fixes in near-term app updates.

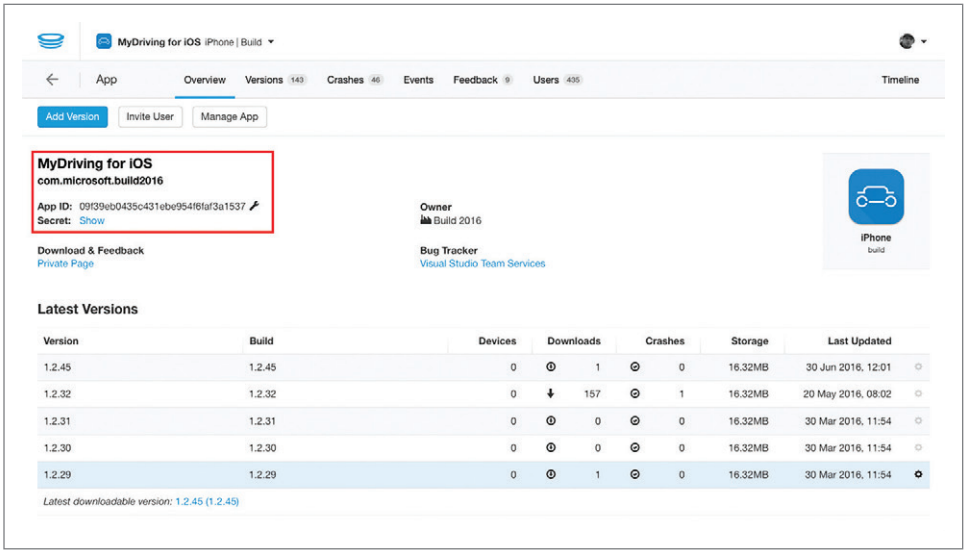


Figure 2 **Finding the App ID on the HockeyApp Portal**

Obtaining information from within running apps means instrumenting the app with HockeySDK. Native SDKs are available for Android, iOS, Windows and macOS; additional SDKs support cross-platform technologies like Xamarin, Cordova, React Native and Unity.

Let's look at a Xamarin app as an example. First, you'll need a unique App ID from HockeyApp for each target platform. Start by creating a free account on hockeyapp.net, then click New App on the main dashboard. This invites you to upload an app package, or click "Create the app manually instead" to enter details directly. Either way, you'll then see a page like the one in **Figure 2** (for the MyDriving iOS app, aka.ms/iotsampleapp), where the App ID appears in the upper left.

For Xamarin, install the HockeySDK.Xamarin NuGet package into each project in the Xamarin solution, except with Windows where you use HockeySDK.UWP. You'll then initialize the SDK during startup using the App IDs from the portal. For iOS, place the following code within the FinishedLaunching method (AppDelegate.cs), replacing APP_ID_IOS with the value from the portal:

```
using HockeyApp;

// ...
var manager = BITHockeyManager.SharedHockeyManager;
manager.Configure("APP_ID_IOS");
manager.StartManager();
manager.Authenticator.AuthenticateInstallation();
```

This code automatically enables crash reporting, user and session tracking, and in-app updates for pre-release builds (those distributed through HockeyApp). To disable features, set properties of the BITHockeyManager class before calling StartManager. For example, the following code disables the in-app update feature:

```
var manager = BITHockeyManager.SharedHockeyManager;
manager.Configure("APP_ID_IOS");
manager.DisableUpdateManager = true;
manager.StartManager();
manager.Authenticator.AuthenticateInstallation();
```

In Android, put code like the following into OnCreate (MainActivity.cs), again using your App ID, and add a call to MetricsManager.Register because session tracking isn't done automatically:

```
using HockeyApp.Android;
using HockeyApp.Android.Metrics;

CrashManager.Register(this, "APP_ID_ANDROID");
MetricsManager.Register(Application, "APP_ID_ANDROID");
UpdateManager.Register(this, "APP_ID_ANDROID");
```

For Windows with the HockeySDK.UWP NuGet package, it's just one line in the App constructor:

```
HockeyClient.Current.Configure("APP_ID_WINDOWS");
```

Complete documentation for the HockeyApp API can be found on support.hockeyapp.net/kb. For Xamarin.iOS and Xamarin.Android, specifically look at "How to Integrate HockeyApp with Xamarin," at bit.ly/2elBqGp. For Windows, see "HockeyApp for Applications on Windows" at bit.ly/2eiduXf. You can always use the MyDriving app code as a reference, too.

With the SDK initialized, let's try out the crash reporting on iOS. The best approach is to throw an exception in a button handler or use the test crash that's part of HockeySDK. In iOS, for example:

```
crashButton.TouchUpInside += delegate {
  var manager = BITHockeyManager.SharedHockeyManager;
  manager.CrashManager.GenerateTestCrash();
};
```

After adding this code (and a button to the UI), build and deploy a *Release* version to a test device, run without debugging, tap the button to trigger a crash, and then launch the app again. A dialog should appear that asks you to submit the crash report, and a few seconds later the crash will appear on the HockeyApp portal. There, you'll see that the crash report shows only memory addresses because the *Release* build doesn't contain debug symbols. But don't enable symbols, because that would allow hackers to easily disassemble your app. Instead, the bin folder of your Xamarin project contains .dSYM (for iOS) and .pdb (Windows/Android) files, which

DOMAINS | MAIL | HOSTING | eCOMMERCE | **SERVERS**

STOP SHARING!

1&1 VIRTUAL SERVER CLOUD

starting at **$4.99** per month*

Trusted Performance.
Intel® Xeon® processors.

Windows

NEW

1&1 eliminates the "noisy neighbor effect": Ideal for beginners as a web and mail server, but also for more demanding projects like database applications, the new 1&1 Virtual Server Cloud is 100% yours to use! Take advantage now of the latest cloud technology.

- No shared resources through VMware virtualization
- Full root access
- SSD storage
- Unlimited traffic
- High performance
- Maximum security
- Best price-performance ratio
- 24/7 expert support
- Choice between Linux/Windows
- Plesk ONYX

1 CALL SPEAK WITH AN EXPERT 24/7

1 MONTH FLEXIBLE PAYMENT OPTIONS

1 SECURE CHOICE FAIL-SAFE

1-844-296-2059

1&1®

1and1.com

* 1&1 Virtual Server Cloud S: $4.99/month. Billing cycle 1 month. Minimum contract term 12 months. No set up fee.

Figure 3 **The HockeyApp Portal Showing Full Symbol Information for a Crash Report**

are standalone bundles of debug symbols. You can upload these symbol files to HockeyApp (called server-side "symbolication") so that crash reports will display full class, method and file names, as well as the line numbers, as shown in **Figure 3**.

The crash overview page shows how often a crash has happened, how many users were affected, and which OS version and device model those users were using. You can compare this to the number of unique users and sessions shown on the app overview page. Look especially at the indicator for the percentage of crash-free users, which is a good measure of app quality.

> Submitting feedback opens a conversation between the user and your team, and once your team replies, the user gets an in-app notification to keep that conversation going.

In addition to these standard metrics, you can get more insights into how your customers use your app by adding custom events. For example, the following line would record when a user starts playing a video:

```
HockeyApp.MetricsManager.TrackEvent("Video started");
```

You'll use the name "Video started" when searching for events in the HockeyApp portal, so it's important to give every event a meaningful name. It's also important to define events at a level that will produce meaningful and actionable data; again, for general guidance refer to the Instrumenting Your App for Telemetry and Analytics blog post at bit.ly/2dZH5Vx. Note that HockeyApp limits you to 300 unique event names per app, but will log unlimited instances of each event name.

To attach more information to an event, use properties like this:

```
HockeyApp.MetricsManager.TrackEvent(
  "Video Started",
  new Dictionary<string, string> { { "filename", "RickRoll.mp4" } },
  new Dictionary<string, double> { { "duration", 3.5 } }
);
```

Although these properties and measurements aren't visible in HockeyApp, they can be explored with the Analytics feature in Microsoft Application Insights, as explained on bit.ly/2ekJwjD.

Another way to understand user behavior it is to directly engage with customers through the HockeyApp feedback feature, which is important for both pre-launch and post-launch customers alike.

When you make a new test release during the pre-launch phase, all your testers will receive an e-mail to which they can reply directly to report issues and suggestions. HockeyApp parses those responses and collects them for you on the portal. Then, for both pre- and post-launch customers, you can integrate a feedback view right in the app using the HockeyApp SDK. Here's how to use the iOS shake event for this purpose:

```
public override void MotionEnded(UIEventSubtype motion, UIEvent event)
{
  if (motion == UIEventSubtype.MotionShake)
  {
    var manager = BITHockeyManager.SharedHockeyManager;
    manager.FeedbackManager.
ShowFeedbackComposeViewWithGeneratedScreenshot();
  }
}
```

The HockeyApp FeedbackManager automatically captures a screenshot of the view that's visible when the shake gesture occurs. It then brings up a view in which the user can annotate the screenshot, attach more screenshots, and provide a text message

to describe the problem or suggestion, as shown in **Figure 4**.

Submitting feedback opens a conversation between the user and your team, and once your team replies, the user gets an in-app notification to keep that conversation going. This is perhaps one of the most valuable features of HockeyApp—putting you in *direct* contact with your customers, which is hard to achieve otherwise. Those direct conversations are a wonderful source of specific, detailed data that can help to validate (or invalidate) your interpretation of the broader data you obtain from telemetry events. Both are really necessary complements.

Like crash reports, user feedback should ultimately create new work items in your development backlog so your team can address issues in future releases. This closes the DevOps loop as shown back in **Figure 1**. In fact, you can connect HockeyApp directly to your backlog by clicking on the Manage App button, then clicking on Bug Tracker and selecting your backlog tool (such as Visual Studio Team Services). Once connected, HockeyApp automatically creates a new work item for every crash group and feedback thread.

Assuming you have a process in place to regularly triage and prioritize new work items, you can respond quickly to crashes and feedback from your customers by making the appropriate changes in your source code. If you've also set up an automated release pipeline as you've seen through this series of articles, those changes will quickly be built, tested and fed into release management so they quickly make their way to customers. And once that release is done, your continuous monitoring should reveal fewer crash reports, higher customer satisfaction and greater overall app performance in every way that matters to your business.

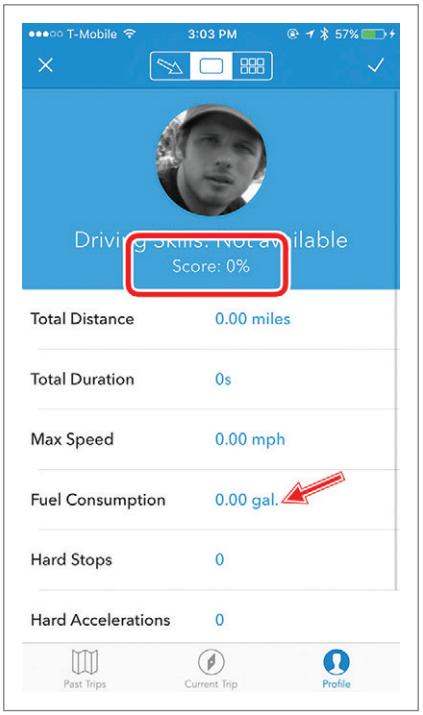


Figure 4 **Annotating Screenshots on iOS**

Monitoring Web Services with Application Insights

Application Insights is a service that gives you a clear view of how your Web apps and services are performing and what your users are doing with them. It notifies you as soon as any problem arises, and provides diagnostic data and tools. It can also analyze and visualize properties of custom events in association with HockeyApp, as described at bit.ly/2ekJwjD. With all that knowledge, you can plan and prioritize future development work with confidence.

Application Insights works for Web apps running on a variety of platforms, including ASP.NET, J2EE, and Node.js, on both cloud and private servers. To instrument an app, start on the Platform Support page (bit.ly/2eydqQU), select your language/platform of choice, and follow the instructions. With a few simple steps, you can then monitor your Web app code, client code in Web pages and any other back-end services, bringing the key data from all components (including HockeyApp) together on a dashboard.

A typical group of performance charts for a Web service, as shown in **Figure 5**, displays information like the following:

- Release annotations (first chart): A blue dot marks deployment of a new app version. In **Figure 5**, notice the slight increase in server response times, which suggests doing some diagnostics to determine why.
- Rate of HTTP requests to the Web service (also the first chart): The thin, lighter segment distinguishes user requests, test requests and requests from search engine bots. Click through for breakdowns by URL, client location and other dimensions.
- Average response time across all requests (second chart): The unusual peak appears to be associated with a spate of exceptions.
- Uncaught exceptions and failed requests (third chart): Shows requests with a response code of 400 or more; click through for individual failures and stack traces.
- Failed calls to dependencies (fourth chart): Shows databases or external components accessed through REST APIs. The peak suggests that some app failures are caused by a failure in another component.

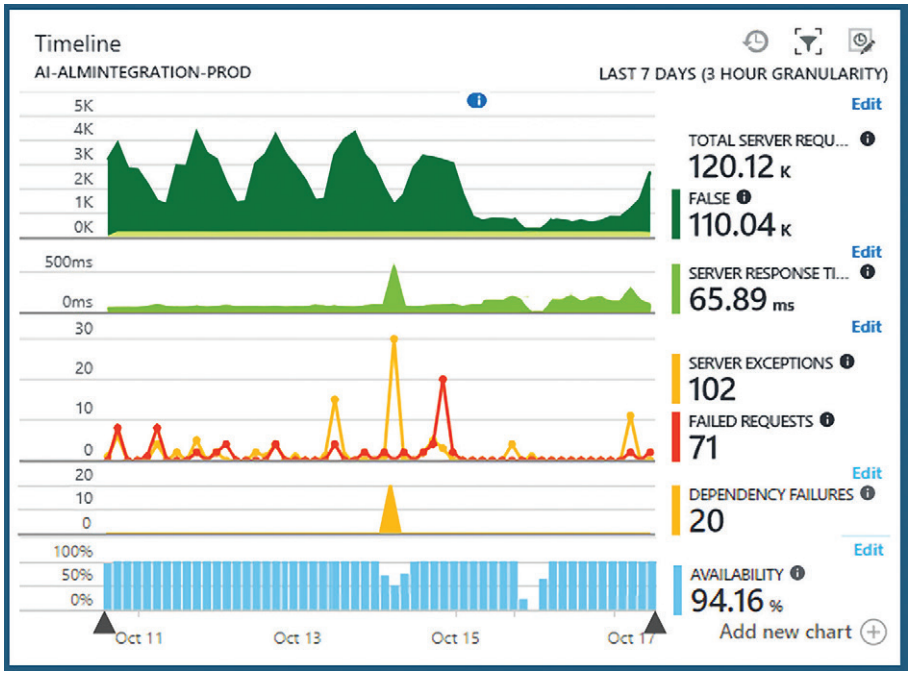


Figure 5 **A Typical Group of Application Insights Performance Charts for a Web Service**

- Availability (fifth chart): Web tests send synthetic requests from around the world to test global availability. The first gap appears to be associated with the sudden rise in exceptions; the second gap might be a network issue or server maintenance.

You can also get charts for a variety of other metrics, such as user, session and page view counts; page load times, browser exceptions and AJAX calls; system performance counters for CPU, memory and other resource usage; and custom metrics that you code yourself.

Of course, you don't want to stare at charts waiting for a failure, so Application Insights can send detailed e-mail notifications for unusual failure rates, availability drops, or any number of other metrics, including those you code yourself. Some alerts are configured automatically, such as smart detection of failures, which learns the normal patterns of request failures for your app, then triggers an alert if the rate goes outside that envelope. In any case, the e-mails contain links to appropriate reports, like that shown in **Figure 6**, through which you can diagnose the problem by clicking through the metric charts and looking at recent events. From a request log, too, you can navigate to the details of any dependency calls, exceptions or custom telemetry events that occurred while the request was being processed.

When you've identified a problem, you'll want to get the issue into your development backlog right away. If you have an automated release pipeline set up for your services, you can make the necessary code changes in short order and quickly test and deploy those changes to your live servers.

To help with this, Application Insights lets you easily create a work item in Visual Studio Team Services directly from a detail view. Just look for the New Work Item + control. Instructions for configuring this connection are at bit.ly/2eneoAD.

You can also consume diagnostics from Application Insights (and HockeyApp!) from directly within Visual Studio using the Developer Analytics Tools plug-in at bit.ly/2enBwIg (a plug-in is also available for Eclipse). For example, every method that's called in the process of handling a Web request is annotated in the code editor with the number of requests and exceptions. There's also an option to instrument the server side of a Web app that's already live, giving you most of the telemetry without the need for access to the code.

With logging, Application Insights captures log traces from the most popular logging frameworks, such as Log4J, Nlog, Log4Net and System.Diagnostics.Trace. These events can be correlated with requests and other telemetry and, of course, you can include custom events and metrics with diagnostic or usage information, both in your back-end code and client code. For example, you can make the following call in a service to periodically monitor the length of an internal buffer, where telemetry is the Application Insights object:

```
telemetry.TrackMetric("Queue", queue.Length);
```

In the Application Insights portal, you can then create a chart to display it, as shown in **Figure 7**.

Other calls can be used to count business or user events such as winning a game or clicking a particular button. Events and metrics can be filtered and segmented by any additional details you send. These events give you insights into what users are doing with your app and how well it's serving them.

If you want to look beyond the regular charts and search facilities that Application Insights provides, you can use Analytics, a powerful query language that you can apply to all the stored telemetry. In **Figure 8**, you can see a query that determines the times of day our Web site is being used, along with the countries that have the most users.

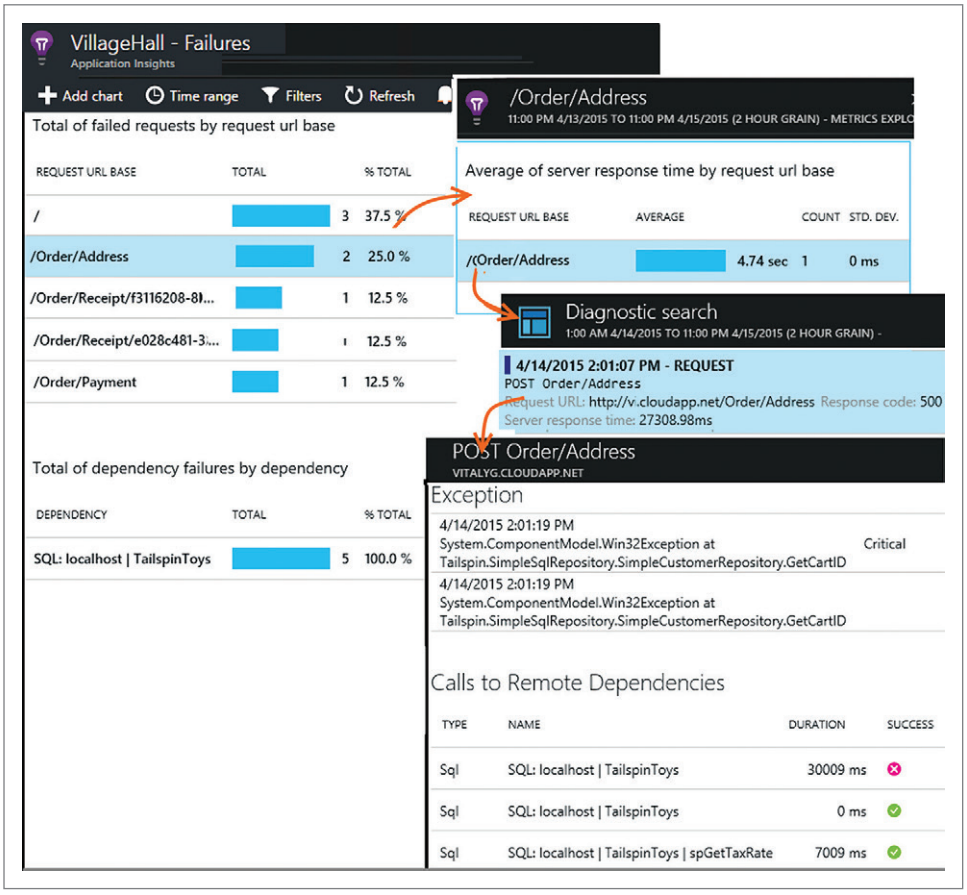


Figure 6 **Clicking Through a Diagnostic Report to See Details**

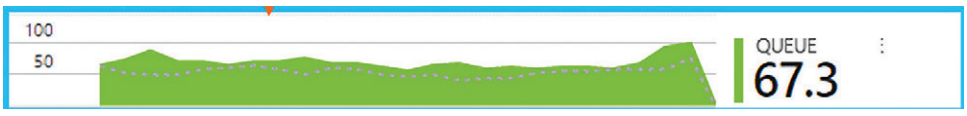


Figure 7 **A Chart in Application Insights for a Custom Metric**

Extreme Performance Linear Scalability

NCache™

Distributed Cache

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

NosDB

NoSQL Database

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

100% Native .NET

Open Source (Apache 2.0 License)

Microsoft Azure amazon web services™ Windows Server 2012 Visual Studio

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

Wrapping Up

We hope you've enjoyed this series of articles that, along with Justin Raczak's article on Xamarin Test Cloud (msdn.com/magazine/mt790199), introduced you to the Microsoft DevOps stack for mobile apps and their associated back-end services. Referring back to **Figure 1**, we've covered all the stages of the release pipeline from the source repository through build, release management, distribution and monitoring. At the same time, we've only scratched the surface: There's much more for you to explore as you deepen the sophistication of your build and release processes, especially where adding multiple levels of automated testing is concerned. The different types of testing that can take place along the release pipeline are again shown in **Figure 1**, and here are some links to get you started:

- **Unit testing** is the process of exercising individual units of code (such as methods). See "Getting Started with Unit Testing for Cross-Platform Mobile Apps" at bit.ly/2ekXt2v, which is a Microsoft Virtual Academy course with Kraig Brockschmidt and Jonathan Carter.
- **Integration testing** also uses unit testing tools but focuses on the interactions *between* units of code (rather than individual units). Integration testing often uses mock data from live servers or databases, or live data in a read-only manner. Running integration tests on real devices using a service like Xamarin Test Cloud is highly recommended.
- For **UI testing**, see Justin's aforementioned article on Xamarin Test Cloud; for Apache Cordova apps, see "UI Testing with Appium" (bit.ly/2dqTOkc) in the Visual Studio Tools for Apache Cordova documentation.
- **Security or penetration testing** means simulating attacks on your apps and services through a variety of vectors, something that can't be taken for granted. We recommend James Whittaker's book, "How to Break Software Security" (amzn.to/2eCbpW4), to get started. This is also an area where you might consider hiring a specialized consultant.
- **Performance diagnostics** are obtained using many of the profiling tools available in Visual Studio itself (see information at bit.ly/2en62c3), along with specialized tools like the Xamarin Profiler (xamarin.com/profiler).
- **Load or stress testing** exercises your back-end services under a variety of conditions to evaluate how they accommodate variable levels of demand. Here, you can also evaluate how your services scale up and down with demand so you can efficiently manage hosting costs. For more, check out Getting Started with Performance Testing at bit.ly/2ey6Dqe, in the Visual Studio Team Services documentation.
- **Localization testing** validates app performance in different locales, which generally means direct UI testing to make sure localized resources and formatting variations are correct. It's also very effective to recruit pre-launch testers around the world because they can point out both technical and cultural issues.

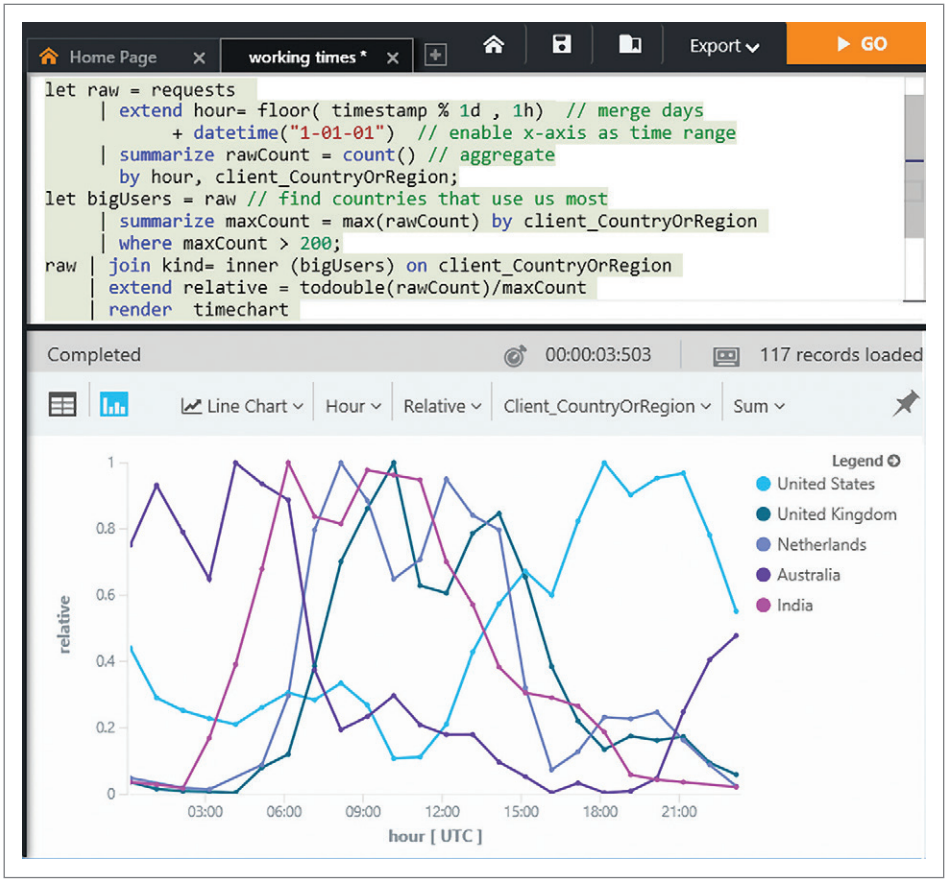


Figure 8 **An Analytics Query in Application Insights**

With that, we again hope you've enjoyed this series, and wish you all the best in your mobile app development endeavors! ■

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

Thomas Dohmke *is a co-founder of HockeyApp, which was acquired by Microsoft in late 2014. At Microsoft, Thomas is a Group Program Manager responsible for driving the product vision and managing a team of program managers for HockeyApp and Xamarin Test Cloud. Reach him via e-mail at thdohmke@microsoft.com or @ashtom on Twitter.*

Alan Cameron Wills *is a senior content developer in Microsoft, writing about Application Insights. He lives and works at Ceibwr Bay, Wales.*

Thanks to the following Microsoft technical expert for reviewing this article: *Simina Pasat*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

7 LOCATIONS TO CHOOSE FROM

JOIN US

ROCK YOUR CODE TOUR 2017

LAS VEGAS
MARCH 13-17

See pages 68-71

AUSTIN
MAY 15-18

See pages 44-45

CHICAGO
SEPT 18-21

WASH. DC
JUNE 12-15

See pages 46-47

ANAHEIM
OCT 16-19

REDMOND
AUG 14-18

ORLANDO
NOV 13-17

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

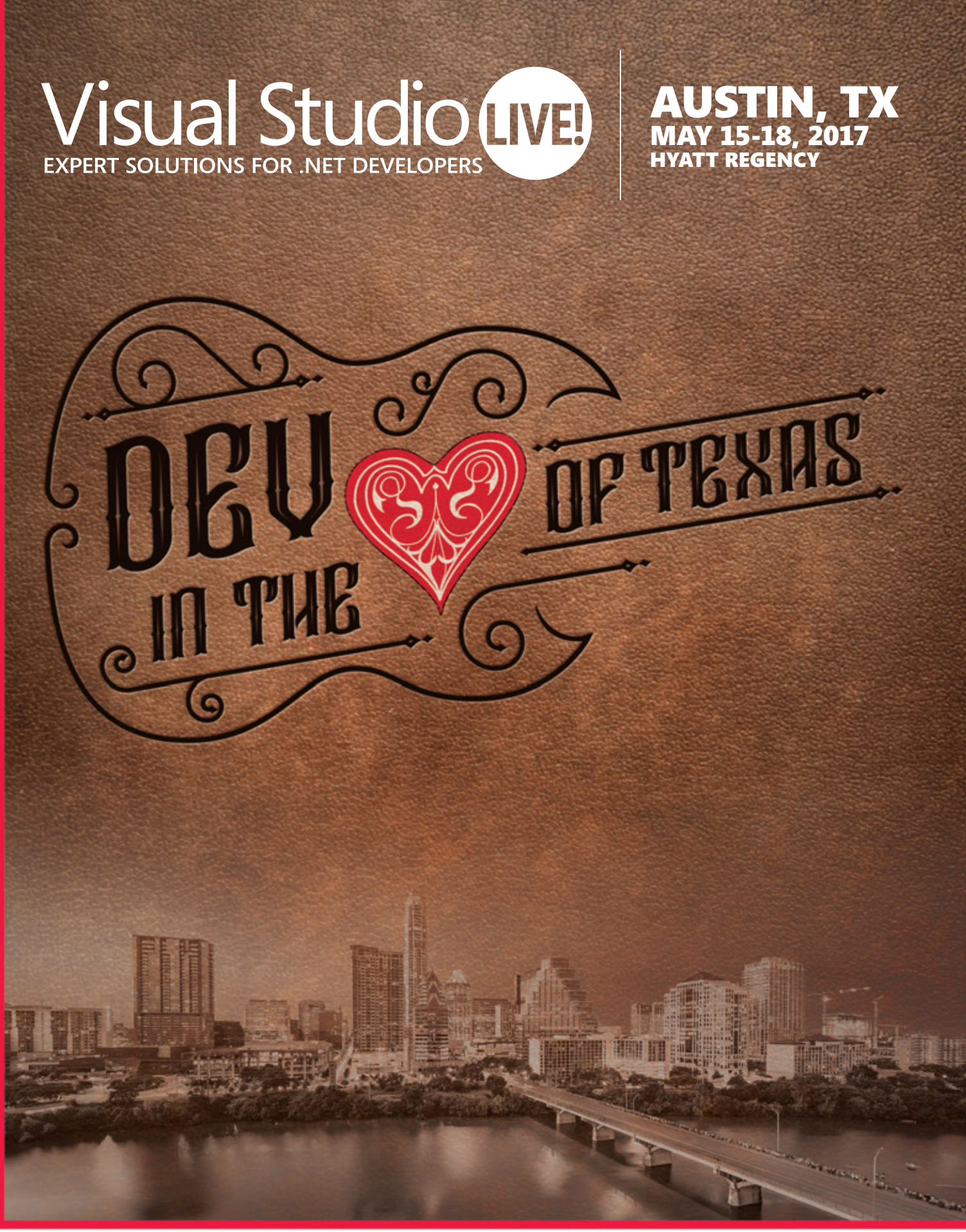
vslive.com

TURN THE PAGE FOR MORE EVENT DETAILS →

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

AUSTIN, TX
MAY 15-18, 2017
HYATT REGENCY



EVENT PARTNER
Microsoft

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

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

Development Topics include:

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

Register by March 17 and Save $300!

Use promo code VSLFEB2

REGISTER NOW

ROCK YOUR CODE TOUR 2017

AUSTIN MAY 15-18

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

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

vslive.com/austinmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

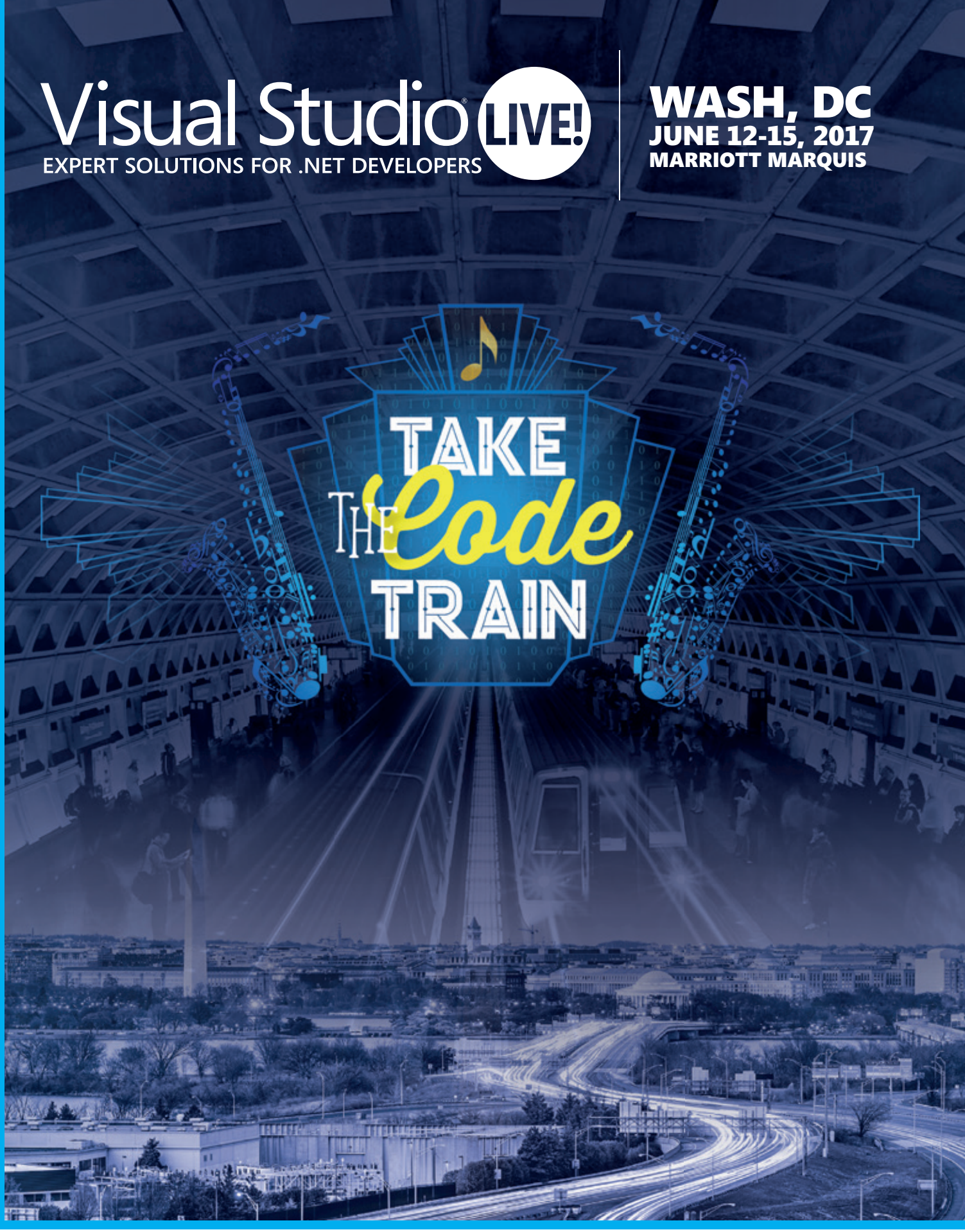
facebook.com – Search "VSLive"

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

WASH, DC
JUNE 12-15, 2017
MARRIOTT MARQUIS



EVENT PARTNER
Microsoft

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

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

Development Topics include:

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

Register by April 21 and Save $300!

Use promo code VSLFEB2

REGISTER NOW

ROCK YOUR CODE TOUR 2017

WASH. DC

JUNE 12-15

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

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

vslive.com/dcmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

Moving from Virtual to Mixed Reality

Tim Kulp

Virtual reality (VR) is all the buzz right now, and developers are racing to build content. On the horizon, mixed reality (MR) is beginning to spark the imagination of developers and make us rethink how the digital world can interact with the real world. As MR gains traction, you'll need to bring your VR apps to a new platform. In this article, I'll explore how to update a VR app to MR on the HoloLens—without having to rewrite the whole thing.

In the September issue of *MSDN Magazine*, I built a VR app called Contoso Travel (msdn.com/magazine/mt763231). This app allowed the Contoso Travel department to see where its employees were traveling throughout the world using a VR map. The idea of the app is to provide an immersive experience for users with an interface that's more than a bunch of pins on a map. In the app, each traveler is represented by an avatar that shows the person's travel destination and times. I used gaze detection to know when the user was looking at an avatar and then allowed the user to select the avatar for a dialog window to show the traveler's details.

This article discusses:

- The difference between virtual reality (VR) and mixed reality (MR)
- Converting an existing VR app to MR
- Using the HoloToolkit to create an MR app
- Building cross platform to support MR and VR from one code base
- Making the app interactive with gaze and gesture
- Implementing voice for input

Technologies discussed:

Unity, Visual Studio, HoloLens, HoloToolkit, Voice and Gesture Input

Code download available at:

bit.ly/2fXiqy2

> When building apps for any platform, it's important to understand what makes that platform unique and embrace it to provide value for that specific platform.

Before You Get Started

This article assumes you've already done some reading on HoloLens development, such as Adam Tuliper's November *MSDN Magazine* article (msdn.com/magazine/mt788624) or the tutorials at the HoloLens Academy (bit.ly/2gzYYr6). Specifically, this article will focus on using gaze, gesture and voice, so you may want to review those articles as I won't go into the details provided in that content. This article

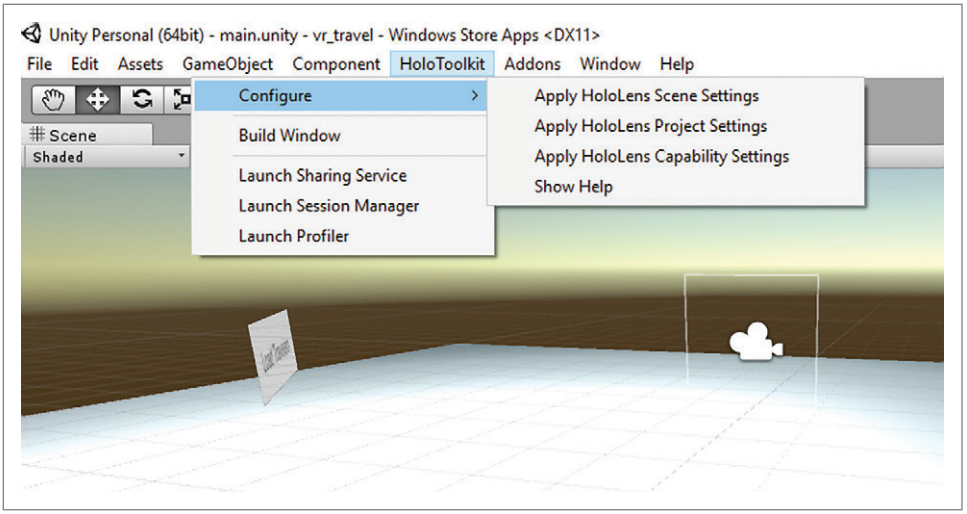


Figure 1 **HoloToolkit Menu**

also assumes you've read my September article as it's the starting point for what I'll cover here.

To start, download the code for vr_travel from GitHub (bit.ly/2fXiqy2). This is the code base I start from in this article.

Virtual Reality vs. Mixed Reality

When building apps for any platform, it's important to understand what makes that platform unique and embrace it to provide value for that specific platform. For example, if there were no difference between a Web application and a mobile application, why would you ever choose one over the other? The key is to provide value based on the different capabilities of each platform. For VR, a key strength is that users only experience what they can see in the headset. The rest of the world fades away as content delivered to the user is exclusive of the real world. This allows for creating amazing worlds that couldn't exist in the real world, or preventing distractions from anything but your app's content. VR takes the user somewhere else. Use it to isolate the user within your experience. From a business application perspective, this could be a call center scenario where all activity done by the user is constrained to the center's operations platform.

MR, in contrast, is inclusive of the world as it combines the virtual world with the real world to create an overall experience. Successful MR uses virtual content to extend the real world. Use MR when you want users to be able to work within your world while still being able to experience theirs. For example, in the office, MR can be used to show content that would always be present, such as a desk calendar connected to Outlook or a modern Rolodex that represents contacts as holographic contact cards. These holograms can be present while users work on real monitors for things like writing software. Engineers can use holograms to see holographic representations of the models they're working on while designing the model on their computers. Office workers could augment their workspace without cluttering it with physical things.

For Contoso Travel, I want users to be able to see where their travelers are at any time. A hologram on the desk is ideal for this as users could do other work on their monitors (such as booking travel), while a quick glance to the side would show where people are at any time. Travel booking systems are complex and not something you'd want to rebuild in VR, but augmenting existing systems with an MR map of travelers is a perfect solution for enabling updates at a glance without occupying a user's physical workspace.

> For VR, a key strength is that users only experience what they can see in the headset.

Preparing for Mixed Reality

HoloLens development typically involves a lot of the same tasks, such as detecting gaze, gestures and voice, no matter what type of app you're building. Fortunately, Microsoft has built a code library that will help you jump into HoloLens development. The HoloToolkit (bit.ly/2b08XrT) provides a great starting point. Download the HoloToolkit and follow the instructions on GitHub to convert it into a Unity package you can import into your HoloLens projects (bit.ly/2ftiOrY).

With HoloToolkit ready to go as a Unity package, open the vr_travel code base and import the HoloToolkit package using Assets | Import Package | Custom Package. Navigate to where you've saved the HoloToolkit package and select it for importing. Unity will display the Import Package dialog, which shows all the different elements of the package. For this article I import everything but the Example projects. If you'd like to include those in your project, feel free as they provide sample code that could be helpful. Click Import when ready to pull the HoloToolkit package into your vr_travel project. Once the import is complete, you'll notice a new folder in your Project view called HoloToolkit, as well as a new HoloToolkit item on the menu bar that gives you access to some useful HoloLens-specific features of Unity, such as automatically applying the settings to the scene or project to run your app on HoloLens (see **Figure 1**).

Updating the JSON Object Library

Depending on the version of the JSON Object library you're using, you might need to update the JSONTemplate.cs file. In your project folder go to JSON/JSONTemplate.cs and update line 19 to the following:

```
FieldInfo[] fieldinfo = obj.GetType().GetFields() as System.Reflection.
FieldInfo[];
```

This update will allow you to compile the project to run on the HoloLens.

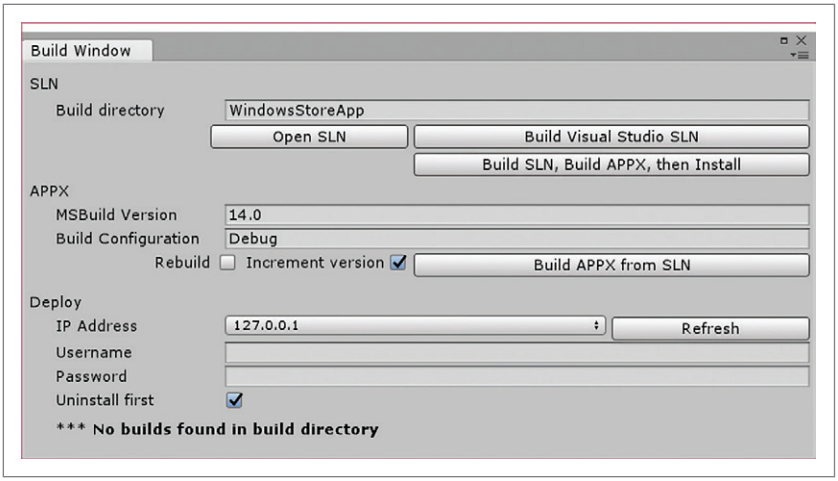


Figure 2 **The HoloToolkit Build Window**

Even though you're updating the app to be a HoloLens app, you might want to keep the VR version of your world. To maintain the VR world, save the main scene as a new scene using File | Save Scene as, and call the new scene main_mr. This way you can maintain the work you've done for the VR world while building a new MR world. Also, as you change the code to components like the Traveler Template or Travel Manager, the VR app will be updated along with the new MR app.

Now, configure your project to be a HoloLens project by going to HoloToolkit | Configure | Apply HoloLens Project Settings. This will update your Build and other project settings to get the app ready for the HoloLens. By configuring the app to be a HoloLens project, the camera will be automatically configured to work on HoloLens. The only change you might want to make from here is to set the Skybox to be a black background instead of using a skybox texture. In the HoloLens, anything pure black isn't rendered as a virtual object. As an example, you can use a black background to make a background transparent, allowing the real world to show through.

> Before you can have the button react to a press, you need to know if the user is looking at it.

With the skybox removed and the project configured, you're left with a surface where the travelers will appear floating in the void. Position the camera to a place where you can view the world, but remember, in a HoloLens app the camera represents the placement of the device. Plan for the camera based on where you want the user's experience to start, and think about how the camera will move throughout the environment. In VR apps, the player is constrained to what the controller allows them to do. For a HoloLens app, you must think through the ways in which the HoloLens can move throughout the world and plan accordingly to provide the best experience possible for users.

With the camera in place, you're ready to test in the HoloLens. Go to File | Build Settings to add the main_mr scene as your primary scene. Click Add Open Scenes to add main_mr, then uncheck the main scene to remove it from the list of built scenes. This allows you to build only the scene that's meant for the HoloLens. You could build from here, but instead, let's use the HoloToolkit build window. Open HoloToolkit | Build Window (see **Figure 2**), which allows you to customize your HoloLens app's build and deployment.

Use the default settings and click Build Visual Studio SLN. This will create a solution for you to open in Visual Studio and deploy to the HoloLens device. When you complete the deployment to the HoloLens Emulator or device, you'll notice you can see the platform and a giant Load Travelers button, but you can't actually do anything yet. Let's make some changes to allow the user to load the travelers. If you need help with building your project, check out "Holograms 100" on HoloAcademy (bit.ly/2bxVOoe) or "Holograms 101" for deploying to a device (bit.ly/2bhqsiV).

Adding Gaze and Gesture

The virtual reality world created in the previous article had some interactivity, so let's reproduce that. The first interactable added to that VR project was the button to load the travelers. Let's make that work for the HoloLens user. To start, create a new Empty Game Object called Manager. (Note: I consolidate all my "managers" to a single object so I have a central place to add components like the Gaze Manager, the Gesture Manager and so forth. This centralization makes it easy to find the managers as scenes get complex, and it simplifies access between manager components.)

With Manager selected, click Add Component and search for Gaze Manager. This enables the camera to provide gaze information to the app so you can detect when the user's view is colliding with an object. Gaze in MR works much like the gaze code you set up for VR. The camera projects a raycast and then detects whether that raycast collides with an interactable object. The Gaze Manager component is prebuilt in the HoloToolkit and can be added as a component right to the Manager object.

A critical UI element for virtual and mixed reality development is to provide visual feedback about what the user is currently looking at. To do this, you need a cursor, and HoloToolkit has one ready to be used. In the Project folder, go to HoloToolkit | Input | Prefabs | Cursor. Drag this prefab onto the Hierarchy. This will add a cursor to the scene that tracks where the user is looking, helping users to not get lost as they look around the scene.

Before you can have the button react to a press, you need to know if the user is looking at it. The button is a Unity Canvas Button that won't react to the raycast because it doesn't have a collider on it. Add a box collider to the btnLoad object by clicking on btnLoad in the Hierarchy and then Add Component. Select Physics | Box Collider and then click Edit Collider. This allows you to resize the collider to that of the button. Set the X scale equal to the width of the button and the Y scale equal to the height of the button. For this article I'm keeping the UI very simple, but there's a lot you can do

Spreadsheets Made Easy.

Fastest Calculations

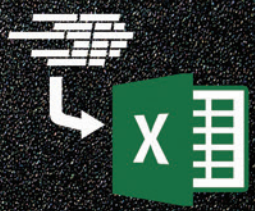
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Powerful Controls

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

Comprehensive Charting

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



Scalable Reporting

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

to create a rich UI for the HoloLens. Check out Surya Buchwald's article, "Scaling UI for the HoloLens" (bit.ly/2gpfGue), for how to set up a scalable UI that the design team can control without needing to engage development.

The button is now ready to react to a gaze. This requires adding the OnGazeEnterEvent and the OnGazeLeaveEvent, which tell the app on what to do when the user is looking at the button as well as when the user looks away. To do this, click on btnLoad again in the Hierarchy and then Add Component. Search for OnGazeEnterEvent and add it. Then do the same for OnGazeLeaveEvent. Once these are added to btnLoad, you can add events. In the Inspector, click the plus sign to add an event to the OnGazeEnterEvent. Drag the EventSystem object from the Hierarchy to the Object field. This will load the functions for EventSystem. In the functions list select EventSystem | SetSelectedGameObject. This controls which game object has focus within the world. Drag btnLoad to the argument object. This code block says when the user's gaze enters btnLoad, trigger the event system to set btnLoad as the selected game object, which will change the button's visual state to that of the highlighted color.

In the OnGazeLeave event, add an event just like the OnGazeEnter event with all the same parameters, but instead of using btnLoad as the argument for SetSelectedGameObject, create a new GameObject in the Hierarchy called UnSelected and set that to be the argument. This will make UnSelected the selected object in the event system when the user's gaze leaves the button. UnSelected is a placeholder object that doesn't do anything but receive the focus when the user looks away from an object. By using this game object you can set which game objects are selected or unselected using the Unity Editor instead of writing code. **Figure 3** shows what the final configuration for btnLoad would look like with the gaze events.

The app now can track where the user is looking and provide feedback to the user. Now let's connect the ability to select the button with an air tap. Click on Main Camera in the Hierarchy; again, click Add Component in the Inspector and search for Gesture Manager. Add Gesture Manager to Main Camera. Gesture Manager allows the app to know when a user's hand is present, recognize which gesture is being used and react to the gesture accordingly.

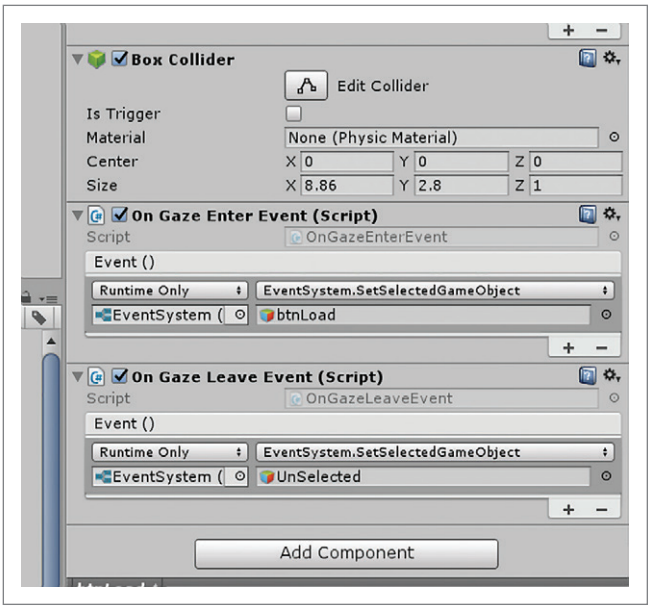


Figure 3 **Configuring btnLoad**

To make a game object react to a gesture, you add the OnSelectEvent component to the game object. Click on btnLoad and then Add Component, search for On Select Event and add the component to the button. Gesture Manager detects when a tap or manipulation gesture occurs and uses GameObject.SendMessage("OnSelect") to trigger the OnSelect event component. The problem with this is that a UnityEngine.UI.Button object already has an OnSelect event (used to activate the button's highlight), so when GestureManager sends the OnSelect message to the button, the button's existing OnSelect event will be triggered. To avoid the conflict of having two OnSelect events, rename the HoloToolkit OnSelectEvent to OnTapEvent. Inside the OnSelectEvent.cs file (under HoloToolkit | Input | Scripts), update the OnSelect method to OnTap and rename the class to OnTapEvent.

> A critical UI element for VR and MR development is to provide visual feedback about what the user is currently looking at.

Another challenge buttons present is that the OnClick event isn't an accessible event through the editor. To trigger a click event, you must call Invoke on the Click event. To do this, create a new script in the Project | Scripts folder called ButtonInteractable, as follows:

```
public class ButtonInteractable : MonoBehavior {
  public void Click()
  {
    var btn = GetComponent<UnityEngine.UI.Button>();
    btn.onClick.Invoke();
  }
}
```

Click on the btnLoad object in the Hierarchy and add the ButtonInteractable component. On the OnTapEvent component, drag btnLoad to the object and then, under function, select ButtonInteractable | Click. This configures the OnTap event to trigger the OnClick event. In this project the button doesn't do a lot with the click event, but in a more complex project a button's click event could trigger many actions. With ButtonInteractable you can maintain the click's behavior without updating it to the OnTap event.

With the Load Travelers button now connected, build the HoloLens app and try it out. When you gaze at the Load Travelers button it turns green. Looking away from the button returns it to its normal state. Using an air tap will load the travelers and deactivate the button so the user can't load the travelers again. Now let's update the travelers to work with gaze and gesture.

Updating the Travelers

The traveler template object is the prefab used to spawn new travelers in the app. On the template are two script components—Traveler Interaction and VR Interactive Item. The VR Interactive Item

receives events to trigger states such as Over (when the user is looking at the interactive item), Out (when the user looks away) and Click (when the user presses the fire button on the VR device). These map directly to Gaze Enter, Gaze Leave and On Tap. Some quick component additions and the travelers will be ready to be used.

On the traveler template prefab add the following components: OnGazeEnterEvent, OnGazeLeaveEvent, and OnTapEvent. As shown in **Figure 4**, for the OnGazeEnterEvent, add the TravelerTemplate object with a function select of VRInteractiveItem | Over. Do the same for OnGazeLeaveEvent except select VRInteractiveItem | Out for the function. This will enable the Over and Out events for the VR Interactive Item. Finally, on the On Tap Event, drag the TravelerTemplate prefab on to the object. For the function, select VRInteractiveItem | Click. This quick configuration brings your VRInteractiveItem into the HoloLens gaze and gesture capabilities.

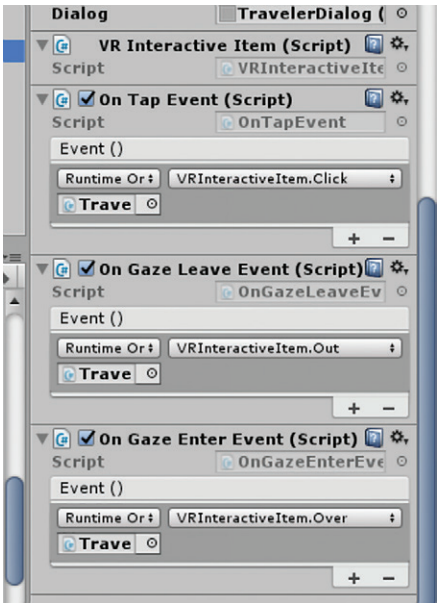


Figure 4 **Adding Components and Functions to the Travelers Template**

At this point, the Contoso Travel HoloLens app has functionality equivalent to the Virtual Reality app. Now you can leverage the features of HoloLens that make it different from a VR device. To start, I'll implement voice so I can load the travelers without clicking a button.

> Voice is an excellent tool to provide users short cuts for actions. Instead of triggering a menu through an air tap, users can just say what they want to do.

Voice as Another Input Mechanism

In traditional VR apps input is limited to gaze, directional pad and a fire button (for a device like Samsung Gear VR). With HoloLens there are multiple other input types, such as voice, which can create a very natural interface for users. Voice is an excellent tool to provide users shortcuts for actions. Instead of triggering a menu through an air tap, users can just say what they want to do. You can make voice commands active only when a specific game object is selected or make a universal command for which the system is always looking.

To start using voice, I'm going to listen for the user to say "Load Travelers," which avoids the need to click the Load Travelers button. A great place to start understanding voice interactions is Adam Tuliper's HoloLens article (mentioned earlier), in which he provides an introduction to implementing voice commands. For this app, I'll apply the voice command by adding a new component to the Manager object. In the Hierarchy, click on Managers | Add Component and then search for KeywordManager. Once the Keyword Manager is added, expand it, set Recognizer Start = Autostart so the recognizer starts with the app. Then expand the Keywords and Responses section and set the size to 1 to add a keyword. Enter "Load Travelers" as the keyword, then connect the function to btnLoad and the ButtonInteractable.Click event. This emulates the button click when the user says "Load Travelers."

You can connect keywords to the travelers, as well. Start by clicking on the TravelerTemplate prefab and deactivate the On Tap event. Add a function to the Gaze Enter event to activate the On Tap event and then add a function to the Gaze Leave event to deactivate the On Tap event. This setup allows only the TravelerTemplate with gaze to display traveler details. Next, update the Keyword Manager to a size of two and make the new entry "Who are you." Then drag the TravelerTemplate to the object and select the function to be VRInteractiveItem | Click. This will open the traveler information dialog.

These minor configuration changes in Unity allow you to enable voice in your mixed reality app. As you think about an office setting where people can be working with keyboards or mice, voice becomes a powerful input device for users with their hands full.

Where to Go from Here

HoloLens and MR offer a lot of potential for your apps. Here, I extended my VR app with voice but I could do even more with spatial sound and mapping. Imagine users binding their travel app to their desk or having the travelers talk to a user when travel plans change. Consider how to mix input systems like gesture, gaze, and voice to provide a rich, fun interface when combined with spatial sound and mapping.

As you can see, VR apps can migrate to MR apps without much code due to the configuration nature of Unity. This saves developers time and money when converting their apps from a mobile VR platform to MR with HoloLens. Take what you've started here and further explore the HoloLens Academy to apply even more features of the HoloLens to your next MR app. ■

Tim Kulp *is the principle engineer at bwell in Baltimore, Md. He's a Web, mobile and Universal Windows Platform app developer, as well as author, painter, dad and "wannabe Mad Scientist Maker." Find him on Twitter: @seccode or via LinkedIn: linkedin.com/in/timkulp.*

Thanks to the following Microsoft technical expert for reviewing this article: *Adam Tuliper*

JAMES MCCAFFREY

The Sign Test Using C#

The sign test is most often used in situations where you have "before and after" data and you want to determine if there's statistical evidence of an effect of some sort. The idea is best explained by example. Suppose you're working at a pharmaceutical company and want to know if a new weight-loss drug is effective. You get eight volunteers to use your drug for several weeks. You look at the weights of your eight subjects before and after the experiment. Say six out of eight of the subjects lost weight. Is there solid statistical evidence to suggest that your drug worked?

Weight loss is a classic sign test example, but the test applies in many IT and software scenarios, too. Suppose you have 40 Web server machines and you apply a software patch designed to improve performance. You measure response times before and after applying the patch. What can you conclude if 32 servers showed better performance, two servers showed no change, and six servers showed worse performance?

The best way to see where this article is headed is to take a look at the demo program in **Figure 1**. After reading this article, you'll have a solid grasp of what type of problem the sign test solves, know exactly how to perform a sign test using C# and understand how to interpret the results of a sign test. All of the source code for the demo program is presented in this article. You can also get the complete demo program in the code download that accompanies this article.

The demo program sets up eight pairs of before-and-after data where the goal is to determine if some weight loss regimen had an effect or not. From the data, six of the subjects did show a weight loss, but two subjects showed a weight increase. The demo program computes the probability of "no effect" to be 0.1445. It's up to you to interpret the results, for example, "The data shows a weak indication (p = 0.8555) that the weight loss effort had an effect."

This article assumes you have at least intermediate programming skill but doesn't assume you know anything about the sign test. The demo code is written in C# and relies on the .NET System.Numerics namespace, so you'll need the Microsoft .NET Framework 4 (released in 2010) or later.

The Demo Program Structure

To create the demo program I launched Visual Studio and selected the C# Console Application template from the New Project menu item. I named the project SignTestUsingCSharp. After the template code loaded into the editor window, I right-clicked on the file Program.cs in the Solution Explorer window and renamed the file to SignTestProgram.cs, then allowed Visual Studio to rename class Program for me.

Next, I right-clicked on the project name and selected the Add | Reference item. From the Assemblies | Framework list, I selected the System.Numerics namespace and clicked OK to add it to my project. At the top of the editor window, I deleted all using statements except for the one referencing the top-level System namespace, and then I added a using statement to reference the System.Numerics namespace.

> The sign test is most often used in situations where you have "before and after" data and you want to determine if there's statistical evidence of an effect of some sort.

The overall structure of the program is presented in **Figure 2**. For simplicity, the program uses a strictly static method approach rather than an object-orienting programming (OOP) approach. Methods DoCounts ansd ShowVector are utility helpers. The work of calculating the no-effect probability is performed by method BinomRightTail. Methods BinomProb and Choose are helpers for BinomRightTail.

After displaying a couple of introductory messages, the Main method sets up and displays demo data for a sign test:

```
double[] before = new double[] { 70, 80, 75, 85, 70, 75, 50, 60 };
double[] after  = new double[] { 65, 78, 72, 87, 68, 74, 48, 63 };
Console.WriteLine("The weight data is: \n");
ShowVector("Before:  ", before, 0, "");
ShowVector("After :  ", after, 0, "\n");
```

In a non-demo scenario with anything larger than about 30 data pairs you'd likely have data stored in a text file, and you'd write a helper method to read and store the data. Using parallel arrays is the most common approach when doing a sign test.

Next, the demo uses method DoCounts to count the number of item pairs where there was a decrease in weight, a "success," and the number of weight increases, a "failure":

```
int[] counts = DoCounts(before, after);
Console.WriteLine("Num success = " + counts[2]);
Console.WriteLine("Num failure = " + counts[0]);
```

Code download available at msdn.com/magazine/0217magcode.

The return value is an array where cell 0 holds the count of fails (weight increase), cell 1 holds the count where there was no change and cell 2 holds the count of successes (weight decrease). In the days before computers were easily available, the counts were done manually by putting a "+" sign next to successes and a "-" sign next to failures. This is why the sign test is named as it is. For the demo data, the manual approach would look like:

```
Before:  70 80 75 85 70 75 50 60
After :  65 78 72 87 68 74 48 63
         +  +  +  -  +  +  +  -
```

Notice the sign test doesn't take into account the magnitude of a weight increase or decrease. Next, the demo prepares the call to the sign test like this:

```
int k = counts[2];
int n = counts[0] + counts[2];
Console.WriteLine("k = " + k + " n = " + n + " p = 0.5");
```

Variable k holds the count of successes. Variable n holds the total count of data pairs. In this situation, there are no instances where the before-and-after weights were equal. In such situations, the most common approach is to toss out ties. However, in some situations you might want to include ties as either successes or failures. For example, in a weight loss program, no change in weight would likely be considered a failure.

The Main method concludes with:

```
double p_value = BinomRightTail(k, n, 0.5);
Console.WriteLine("\nProbability of 'no effect' is " + p_value.ToString("F4"));
Console.WriteLine("Probability of 'an effect' is " + (1 - p_value).ToString("F4"));
```

The sign test is actually a specific example of the more general binomial test. Program-defined function BinomRightTail accepts the number of successes, the number of data pairs and a probability value, 0.5 in this case. When a binomial test uses 0.5 for the probability parameter, it's a sign test, as I'll explain shortly.

Understanding the Choose Function

The sign test uses the binomial distribution, which in turn uses the Choose function. The Choose(n, k) function returns the number of ways to select k items from n items. For example, Choose(5, 3) is the number of ways you can select three items from five items. Suppose the five items are (A, B, C, D, E). There are 10 ways to select three of the items:

```
(A, B, C), (A, B, D), (A, B, E), (A, C, D), (A, C, E),
(A, D, E), (B, C, D), (B, C, E), (B, D, E), (C, D, E)
```

The Choose function is defined Choose(n, k) = n! / [k! * (n-k)!] where the "!" character means factorial. So:

```
Choose(5, 3) = 5! / (3! * 2!) = (5 * 4 * 3 * 2 * 1) / (3 * 2 * 1) *
  (2 * 1) = 120 / 12 = 10
```

Implementing the Choose function is tricky because the return value can be astronomically large for even moderate values of n and k. For example:

```
Choose(100, 25) = 242,519,269,720,337,121,015,504
```

In order to return the very large values that can occur in the sign test, the demo program uses the BigInteger type in the System.Numerics namespace. The demo implementation of Choose uses two math tricks for efficiency. First, as it turns out, Choose(n, k) = Choose(n, n-k). For example:

```
Choose(10, 7) = Choose(10, 3)
```

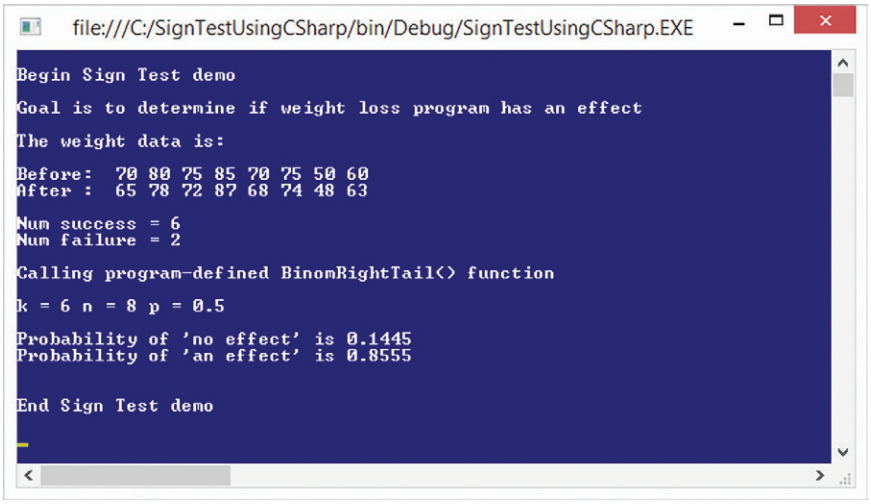


Figure 1 **The Sign Test Using C#**

By using the smaller value of k you can do fewer calculations. Second, there's an alternative definition of Choose that's best explained by an example:

```
Choose(10, 3) = (10 * 9 * 8) / (3 * 2 * 1)
```

In words, the denominator is just k! and the numerator uses just the first k terms of the n! equation and many terms cancel out. Putting these ideas together, the demo implementation of Choose is presented in **Figure 3**.

Understanding the Binomial Distribution

The key to understanding how to implement and interpret the sign test is understanding the binomial distribution. It's best explained by example. Imagine you have a biased coin where, when flipped, the probability of getting a head is 0.6 and the probability of getting a tail is 0.4, and suppose you define a success as getting a head. If you flip

Figure 2 **Sign Test Demo Program Structure**

```
using System;
using System.Numerics;
namespace SignTestUsingCSharp
{
  class SignTestProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("\nBegin Sign Test demo \n");
      // All calling statements go here
      Console.WriteLine("\n\nEnd Sign Test demo \n");
      Console.ReadLine();
    }

    static int[] DoCounts(double[] before,
      double[] after) { . . }

    static void ShowVector(string pre, double[] v,
      int dec, string post) { . . }

    static double BinomProb(int k, int n,
      double p) { . . }

    static double BinomRightTail(int k, int n,
      double p) { . . }

    static BigInteger Choose(int n, int k) { . . }
  }
}
```

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

AUGUST 7 - 11, 2017

MICROSOFT HEADQUARTERS

REDMOND, WA

PLUG IN TO NEW KNOWLEDGE @ THE SOURCE

WHAT SETS TECHMENTOR APART?

+ Immediately usable IT education
+ Training you need today, while preparing you for tomorrow
+ Zero marketing-speak, a strong emphasis on doing more with the technology you already own, and solid coverage of what's just around the corner
+ Intimate setting, where your voice is heard, making it a viable alternative to huge, first-party conferences
+ Experience life @ Microsoft Headquarters for a full week

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

HOT TRAINING TOPICS INCLUDE:

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

++

REGISTER NOW

SAVE $400
USE PROMO CODE TMFEB1

[TECHMENTOREVENTS.COM/REDMOND]

SUPPORTED BY: Redmond · Redmond Channel Partner · Virtualization Review

PRODUCED BY: 1105 MEDIA

Figure 3 **The Choose Function**

```
static BigInteger Choose(int n, int k)
{
  if (n == k) return 1; // Required special case
  int delta, iMax;

  if (k < n - k) { // Ex: Choose(100,3)
    delta = n - k;
    iMax = k;
  }
  else { // Ex: Choose(100,97)
    delta = k;
    iMax = n - k;
  }

  BigInteger ans = delta + 1;
  for (int i = 2; i <= iMax; ++i)
    ans = (ans * (delta + i)) / i;

  return ans;
}
```

the coin n = 8 times, the binomial distribution gives you the probability of getting exactly k successes in n trials where the probability of a success in a single trial is p = 0.6 in this example.

The probability of getting exactly eight heads and zero tails in eight flips is the probability of getting eight consecutive heads, which is:

```
Pr(X = 8) = 0.6 * 0.6 * 0.6 * 0.6 * 0.6 * 0.6 * 0.6 * 0.6 = (0.6)^8 * (0.4)^0 = 0.0168
```

To get exactly seven heads in eight flips you can get seven heads plus one tail on any of the eight flips. There are eight combinations:

```
Pr(X = 7) = Choose(8, 1) * [ (0.6)^7 * (0.4)^1 ] = 8 * 0.0280 * 0.4 = 0.0896
```

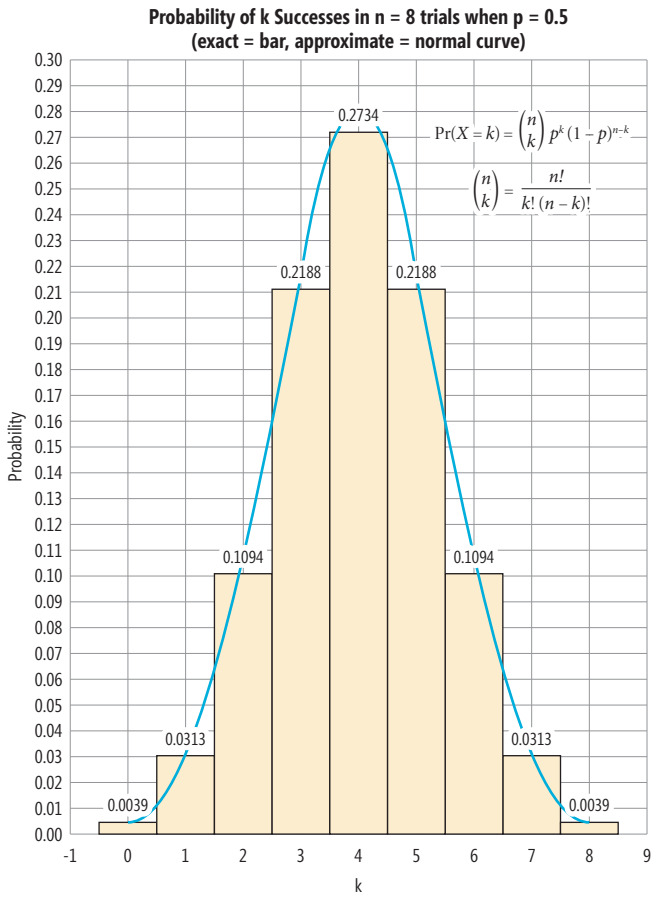


Figure 4 **The Binomial Distribution for n = 8 and p = 0.5**

The general equation for the probability of getting exactly k successes in n trials where p is the probability of a success in a single trial is:

```
P(X = k) = Choose(n, k) * p^k * (1-p)^n-k
```

In the case of the sign test, p is always 0.5 so 1-p is also 0.5 and the equation simplifies to:

```
P(X = k) = Choose(n, k) * (0.5)^n
```

So, for the demo data, there are n = 8 trials (pairs of data) and there are k = 6 successes (weight losses), so the probability of getting exactly six successes is:

```
P(X = 6) = Choose(8, 6) * (0.5)^8 = 28 * 0.0039 = 0.1094
```

The probabilities of getting exactly zero through eight successes in eight trials when p = 0.5 are shown in the graph in **Figure 4**.

Implementing a function that returns the binomial probability is straightforward:

```
static double BinomProb(int k, int n, double p)
{
  // Probability of k "successes" in n trials
  // if p is prob of success on a single trial
  BigInteger c = Choose(n, k);
  double left = Math.Pow(p, k);
  double right = Math.Pow(1.0 - p, n - k);
  return (double)c * left * right;
}
```

The demo defines a general binomial function that accepts p as a parameter. An alternative is to define a version that assumes p = 0.5 and simplify the calculation as described earlier. The demo has no error checking. For example, in a production environment you'd likely want to make sure k <= n; neither k nor n are negative; and p is between 0.0 and 1.0.

Implementing the Sign Test

The idea of the sign test is to calculate the probability that there's been no effect. Conceptually this means any difference between a before-value and an after-value has happened purely by chance. Mathematically, this means that the probability of an increase or decrease is 0.5.

The sign test assumes there's no effect, then calculates the probability that the observed number of successes could've happened under this assumption. For the case of the demo data where there were six successes (weight losses) in eight trials, rather than calculate the probability of exactly six successes as you might guess, you calculate the probability of six or more successes. This idea is rather subtle.

Calculating the probability of k or more successes is sometimes called a right-tail test. So to implement the sign test, you calculate the probability of k or more successes by calculating the probability of exactly k successes plus k+1 successes, plus k+2 successes and so on. The demo implements this as:

```
static double BinomRightTail(int k, int n, double p)
{
  // Pobability of k or more successes in n trials
  double sum = 0.0;
  for (int i = k; i <= n; ++i)
    sum += BinomProb(i, n, p);
  return sum;
}
```

All that's needed to complete a sign test are optional functions to count the number of successes and to display values. The demo defines the counting method as:

```
static int[] DoCounts(double[] before, double[] after)
{
  int[] result = new int[3];
  for (int i = 0; i < before.Length; ++i) {
    if (after[i] > before[i])
      ++result[0];  // Fail
    else if (after[i] < before[i])
      ++result[2]; // Success
    else
      ++result[0]; // Neither
  }
  return result;
}
```

The helper display method is:

```
static void ShowVector(string pre, double[] v,
int dec, string post)
{
  Console.Write(pre);
  for (int i = 0; i < v.Length; ++i)
    Console.Write(v[i].ToString("F" + dec) + " ");
  Console.WriteLine(post);
}
```

An alternative design is to combine the success-failure counting and binomial calculations into a larger meta-method.

Wrapping Up

You should always interpret the results of a sign test cautiously. It's better to say, "The sign test suggests that there is an effect," rather than, "There is an effect."

The example problem is called a one-sided, or one-tailed, test. Because this example involved a weight-loss experiment, an effect is more weight losses (successes) than you'd get by chance. You can also perform a two-sided, also called two-tailed, sign test. For example, suppose you're doing an experiment with some sort of pain medication. As part of your experiment, you weigh your test subjects before and after the experiment. You have no reason to believe that the pain medication will affect weight. In other words, an effect would be either a weight loss or a weight gain.

The trickiest part of the sign test is keeping your definitions clear. There's potential confusion because there are multiple symmetries in every problem. You can define a success as an increase or decrease in an after-value. For example, in the weight-loss example, a success is a decrease in the after-value. But if your data represents test scores on some kind of exam before and after studying, a success would likely be defined as an increase in the after-value.

The sign test is an example of what's called a non-parametric statistical test. This means, in part, that the sign test does not make any assumptions about the distribution of the data being studied. Instead of using a sign test, it's possible to use what's called a paired t-test. However, the t-test assumes that the population data has a normal (Gaussian, bell-shaped) distribution, which would be almost impossible to verify with a small data set size. Because of this, when I want to investigate before and after data, I'll usually use a sign test instead of a paired t-test. ■

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

Thanks to the Microsoft technical experts who reviewed his article: *Chris Lee and Kirk Olynyk*

TED NEWARD

How To Be MEAN: Working the Angular

Welcome back, MEANers. As mentioned last month, it's time to start working the front end of the MEAN stack, which means diving into Angular. As of this writing (and probably for the next couple of years), that presents a problem—Angular has a bit of an identity problem. Angular has been the staple of the Single-Page Application world for many years. However, as Angular developed, it grew into a different architectural paradigm, and as a result, the "modern" Angular grew to become backward-incompatible with Angular.

That leaves most developers with the thorny question: Which version to use?

While there's never a one-size-fits-all answer to the problem, history has shown that with only a few exceptions, the later version eventually wins out. This leads to a basic rule of thumb: If you're just starting to work on the project (a so-called "greenfield" project), and there's no existing code to maintain or extend, then use the latest-and-greatest stable version of the framework. Because my application (the tiny speaker ratings portal that I've been toying around with) definitely falls under the category of greenfield, that means that within the pages of this column series, I'll use Angular 2 for the modern version of Angular and Angular 1 for the original product.

Yes, the decision could have easily gone the other way. Fortunately, the Internet holds dozens of good Angular tutorials (and good on ya, mate, if you choose Angular 1 for keeping all those Angular 1 tutorial writers happy). However, be warned that "porting" an Angular 1 app to Angular 2 looks to be more of a complete rewrite, so be sure to take that into consideration in future plans.

Meanwhile, we have some Angular 2 to explore.

Getting Started

The first step in working with any new technology is to write the ubiquitous "Hello World" for it. Two interesting things emerge immediately, though, when looking to do an Angular 2 "Hello World" app that bear discussing before going too far into the process.

First, with a Web framework like Angular, usually the installation process is almost ridiculously lightweight (compared to installing new programming languages, IDEs, databases and so on), because most of the time the actual library itself can be fetched directly out of a CDN or from the host server. However, for most development tasks, it's better to have the library running off the local filesystem, and because that's the default way to get started with Angular 2, that's the path I'll take.

Second, Angular 2 chooses to make its debut through a Git repository—in other words, the default "getting started" approach is to clone an existing Git repository on GitHub, as opposed to an IDE-hosted "project template." This is an approach that's starting to gain steam with other languages and other frameworks. It has the distinct advantage of being easy to understand, is trivial to maintain and (perhaps most important) is simple to extend to include additional features (structure, contents and so on) that the original template doesn't have.

Thus, assuming Node.js is installed on your machine (which it should be for readers who've been following along in this series), getting the Angular 2 "quickstart" project is a Git request:

```
git clone https://github.com/angular/quickstart.git hello
```

Assuming this connects to GitHub and successfully clones the project, a ton of files now rest in the "hello" subdirectory. There are far more, in fact, than would seem absolutely necessary for a simple "Hello World" application, and the Angular team admits as much. In the repository's README file, they specifically state that they're laying out a number of additional files to promote some good habits right from the beginning, such as unit and end-to-end (e2e) testing of the front-end code.

The first step, of course, in working with any new technology is to write the ubiquitous "Hello World" for it.

Examining much of that will come later. For now, a glance through the directory will reveal a couple of things that should be familiar to MEANers: package.json (the npm manifest file) and tsconfig.json (the TypeScript configuration file) stand out immediately. Recall that the standard practice when pulling a Node.js project out of source control is to boot strap the dependencies into place, so before doing anything else, pull down the dependencies and give the project a wake-up call by executing the following, then browse to port 3000 in a browser (Actually, the script will usually open one for you once a local HTTP server is running on that port.):

```
npm install
npm start
```

Then, bask in the warm greetings of a Web framework, as shown in **Figure 1**.

In and of itself, it's always nice to know that everything works, but programmers want to see code. Head back to the command-line shell that's running the HTTP server and hit Ctrl+C to shut everything down. (Alternatively, open a second command-line shell to the same directory, whichever is easier.)

Let's take a look, shall we?

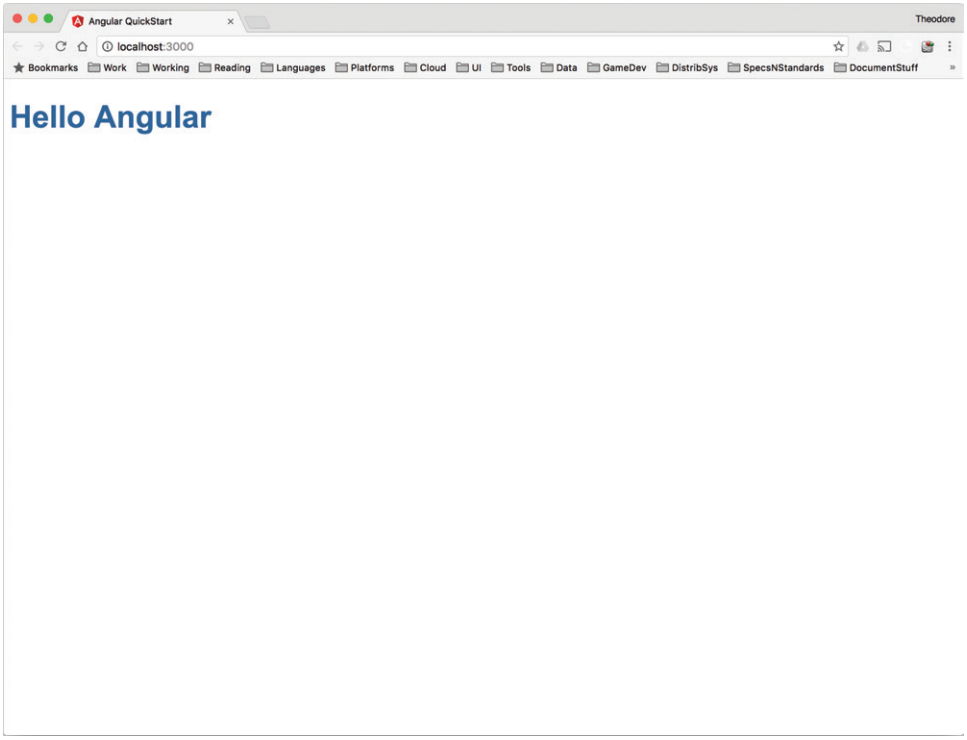


Figure 1 **Hello Angular 2 Web Framework**

Hello, Code

Of course the first place you can look to find code in an Angular 2 application is the index.html file, but this is actually going to be a little more confusing than helpful right now; for the moment, let's leave it alone and dig around elsewhere.

The Angular team will be the first to admit that the directory structure of the Quickstart isn't designed to be a prescriptive guide on how to structure code, but usually all Angular 2 apps will have some sort of "source" directory in which the application resides off the main root of the project. (This makes it much easier to bundle up without pulling in all sorts of developer-only files, such as package.json.) In the Quickstart, this directory is called "app," and it contains three files: main.ts and two files that seem closely related—app.component.ts and app.module.ts. (Note that the TypeScript transpiler will modify these files in-place, so the directory might contain more than just these files, but it'll be a little obvious that they're all related—for example, main.ts will generate main.js and main.js.map). The first, main.ts, is clear as to its purpose—it's the main entry point for the Angular 2 application—but the other two are a little less so. Nevertheless, let's look at all three.

> Like its predecessor, Angular 2 cares a great deal about modularizing the application code into manageable bite-size chunks.

Entry Point: Main.ts

The contents of main.ts are a little cryptic at first:

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app.module';
platformBrowserDynamic().bootstrapModule(AppModule);
```

That's not exactly clear. However, let's take it piece by piece. You know from my last column on TypeScript that the "import" statement pulls in symbols from another file, and you can see that it's importing symbols from two different places. The first, platformBrowserDynamic, is coming from somewhere in the Angular libraries, so this is probably standard boilerplate to boot strap the Angular 2 environment and library. (The last line definitely bears this out.) The second, however, imports from a local file, app.module, which sounds suspiciously like it's supposed to contain your code.

For the most part, main.ts remains untouched throughout Angular 2 development with anything application-related at least living in the module file (app.module.ts), but it's always useful to understand the context. (Although it's not recommended at this point, if you follow the trail from index.html, you'll eventually find where main.js gets loaded through the System.js module-loader mechanism.) That means, then, that most of the action takes place in app.module and its relations.

The Application Module: App.module.ts

Like its predecessor, Angular 2 cares a great deal about modularizing the application code into manageable bite-size chunks, and the first step to doing that is to put application-wide elements into a single place, which in Angular 2 speak is a module. Hence, this file will pull in a few Angular 2 concepts and then declare the application module and what it, in turn, uses:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';

@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Again, for the moment, this primarily is all import statements: First, pulling in some Angular 2-related materials (NgModule and BrowserModule), then importing an application component, which I'll touch on in a second. Notice, however, that NgModule is (as discussed in my last column) a TypeScript decorator—in essence, this is what lets Angular 2 have all the functionality it's expecting while letting developers use the framework to stay focused on just the application-specific features on the application-specific class (which is called AppModule here).

It's actually quite important to understand that Angular 2 takes a strong stance on the segregation of code and features into modules and components. This will be a theme that's repeated over and over again in Angular 2, whereas in Angular 1 it was possible to think about code being arranged more or less in whatever manner the developer chooses (or, more often than not, chooses not to organize at all). In Angular 2 the library forces developers to face the organization scheme right away. Angular 2 lets you choose the granularity of your modules and components to be as large or as small as you like, but you must organize them into modules and components, without question. This is simply "The Angular Way," and you must follow the process.

As the application scales up in size, the overhead will proportionally shrink, and leave application developers to focus entirely on the "meat" of their application. And that's exactly what an application framework is supposed to do.

The NgModule decorator provides metadata about this module, including what other modules it depends on, what declarations it exports (which you'll see used in a second) and what bootstrapping needs to take place. NgModule has several options that you can pass here, and as an Angular 2 application grows in complexity, these lists grow. Bookmark this file some place as you'll be back here often.

A Hello World Component: App.component.ts

The last bit to discuss is the actual application component—the only one, so far—that defines the UI (all one line of it, anyway). This is the app.component.ts file:

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`,
})
export class AppComponent  { name = 'Angular'; }
```

Again, the component depends on an Angular 2 construct—the Component decorator—so it imports it from the appropriate place inside the Angular libraries. It then declares an exported class, called AppComponent, to be an Angular component, with two parameters: a selector and a template. The template is straightforward: This is the HTML snippet applied where this component is declared (complete with ECMAScript string interpolation binding, in this case for the "name" parameter in the HTML). Because this template can sometimes get to be a bit large, or at least larger, than the single line of HTML defined here, instead of "template" you can also use templateUrl to specify an external file in which to find the template, which you'll do in later incarnations.

The selector parameter is a bit more subtle; it declares where in the UI this component is to apply. Practically speaking, this means that anywhere a my-app tag shows up in HTML, this component is applied instead. You haven't seen any <my-app> tags thus far, but that's simply because this particular tag is declared inside of the index.html file (which I haven't discussed):

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular QuickStart</title>
    <!-- bunch of other stuff snipped for simplicity -->
  </head>

  <body>
    <my-app>Loading AppComponent content here ...</my-app>
  </body>
</html>
```

Notice how my-app tag surrounds some text; this primarily is placeholder text that might or might not appear, depending on how quickly the browser loads and renders the application.

Wrapping Up

This has been a lot of work for a simple "Hello World"; it would seem a lot easier to simply write straight HTML and leave out the rest of it. However, a large part of "The Angular Way" is to build the application out of components, rather than just to sling HTML and CSS and JS everywhere, and that kind of organization and structure carries overhead with it. In fact, almost 90 percent of what I've discussed so far is essentially Angular overhead. As the application scales up in size, the overhead will proportionally shrink, and leave application developers to focus entirely on the "meat" of their application. And that's exactly what an application framework is supposed to do.

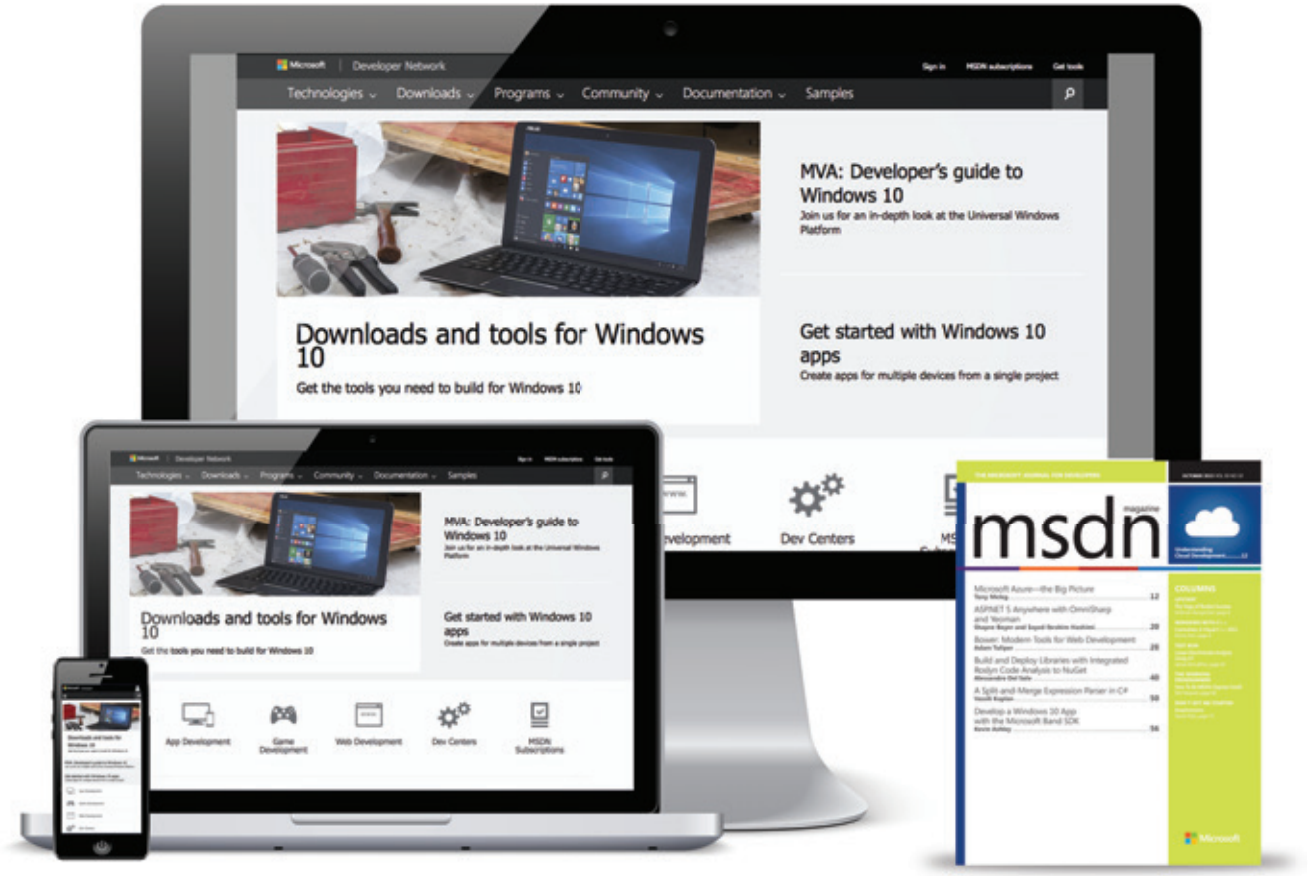
Naturally, however, we're not done yet. There's a lot more of Angular 2 to explore, and we have a barebones application to build, to boot. In future columns, I'll explore creating components, specifically doing some basic CRUD around the list of speakers (using an in-memory "database" to start), and how Angular 2 can make it simple to keep everything straight. Hang tight; much more is yet to come. Until then, happy coding! ■

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

Thanks to the following technical expert for reviewing this article: *Ward Bell*

msdn magazine

Where you need us most.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

Twitter-Searching Utility

In my column last month, I explored the great Universal Windows Platform (UWP) Community Toolkit, an open source toolkit built by the community for the community. However, I barely scratched the surface of what it can do. The UWP Community Toolkit makes building highly polished, cloud-powered UWP apps much easier and faster. In this month's column, I'll discuss how to build a Twitter search app using the Twitter services and the blade control to demonstrate how easy it is to work with.

Currently, I manage several YouTube channels that source content from tweets marked with certain hashtags. For example, #DCTech Minute focuses on the happenings in the DC Area startup and technology scene. I find content to highlight based on tweets that use the DCTech hashtag. For #Node.js Minute, I do the same with tweets marked with #Node.js. Currently, I do a lot of manual cutting and pasting between Twitter and OneNote. It would be great to have a UWP app that can search Twitter for all the key phrases I need all in one window and make it easier to pull the content from the tweets.

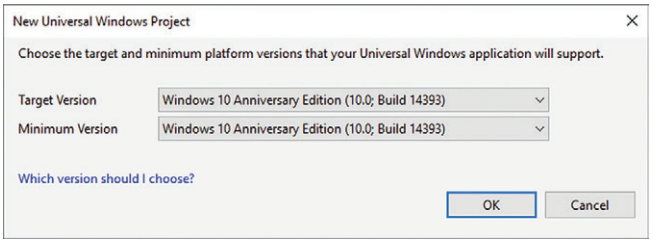


Figure 1 **Targeting the Correct Version of Windows**

Setting up the Project

Create a new blank UWP project in Visual Studio by choosing New Project from the File menu. Expand the Installed Templates | Windows | Blank App (Universal Windows). Name the project TagSearcherUWP and then click OK. Immediately afterward, a dialog box will appear asking you which version of Windows the app should target. At a minimum, you'll need to choose Windows 10 Anniversary Edition (10.0; Build 14393). This is the most recent version. Therefore, both the Target Version and the Minimum Version will both target the same version, as shown in **Figure 1**. If you don't see this particular version in either dropdown list, then make sure you have the appropriate software installed on your system. Failure to select the correct version will yield a runtime error once the Microsoft.Toolkit.Uwp.UI.Controls NuGet package is added to the project.

Once the solution loads, browse to Solution Explorer, then right-click on References and choose Manage NuGet Packages from the context menu to show the NuGet Package Manager. In the search box, type "Microsoft.Toolkit.Uwp" to bring up all the NuGet packages associated with the UWP Community Toolkit. This project will use the Microsoft.Toolkit.Uwp.Services and Microsoft.Toolkit.Uwp.UI.Controls packages. Install them both to add them to the project. If prompted with a Review Changes dialog, review the changes and

Code download available at bit.ly/tagsearcher.

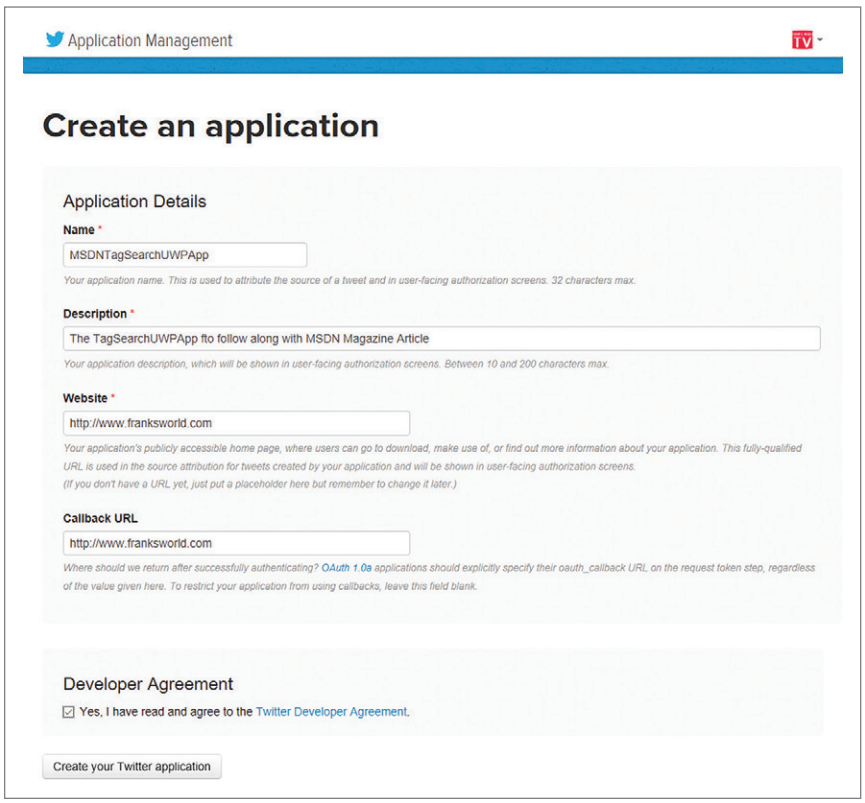


Figure 2 **Creating a New Twitter App**

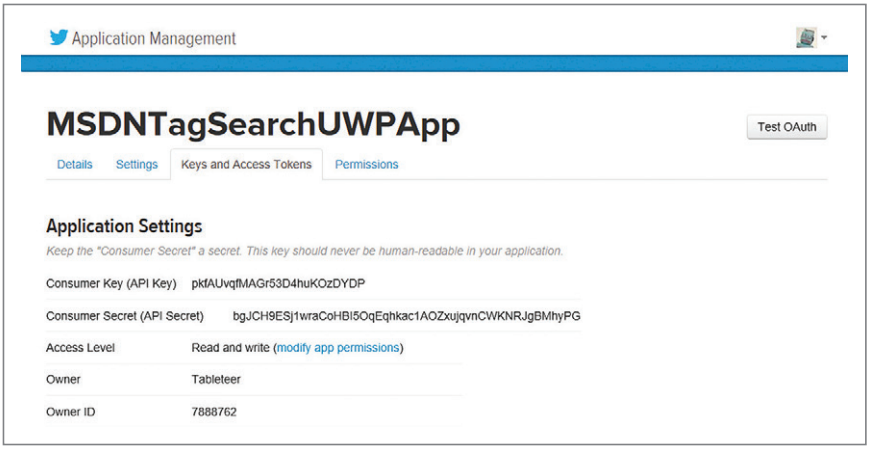


Figure 3 **The Consumer Key and Access Token Tab**

then click OK to accept. You'll also see a License Acceptance dialog for each package. Click "I Accept" to accept the license terms. Clicking "I Decline" will cancel the install.

Setting Up Twitter

Now that the project is set up with the appropriate NuGet packages, it' time to connect the app to the Twitter service. Go to apps.twitter.com and sign in with your Twitter account. If you don't have a Twitter account, you should make one now. If you haven't created a Twitter App, you'll need to click on Create New App to register a new app.

You'll need to fill out details about the app, such as the Name, Description, Web site and Callback URL. You can fill in the fields as you wish. For the name, I chose MSDNTagSearchUWPApp. End users will see the Description text when they log in, so it's best to make it short and descriptive. See **Figure 2** for guidance.

Figure 4 **XAML Code to Create the Interface**

```
<Page
  x:Class="TagSearcherUWP.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:TagSearcherUWP"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:controls="using:Microsoft.Toolkit.Uwp.UI.Controls"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d">
  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
      <RowDefinition Height="56*"/>
      <RowDefinition Height="35*"/>
      <RowDefinition Height="549*"/>
    </Grid.RowDefinitions>
    <StackPanel Orientation="Horizontal" VerticalAlignment="Center">
      <TextBlock FontSize="24" Margin="5">Twitter Tag Searcher</TextBlock>
      <Button Name="btnLogin" Click="btnLogin_Click" >Log In</Button>
    </StackPanel>
    <StackPanel Name="splSearch" Grid.Row="1"  Orientation="Horizontal"
      VerticalAlignment="Center" Visibility="Collapsed">
      <TextBox Name="txtSearch" Margin="5,0,5,0"  MinWidth="140" Width="156" />
      <Button Name="btnSearch" Click="btnSearch_Click">Search</Button>
    </StackPanel>
    <controls:BladeView Name="bladeView" Grid.Row="2" Margin="12"
      HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
      <controls:BladeItem x:Name="DummyBlade" IsOpen="False" />
    </controls:BladeView>
  </Grid>
</Page
```

For both the Web site and Callback fields, I put in my Web site URL. In the case of UWP apps, the callback URL doesn't have to be a working URL. Make note of the URL as you'll need it later when logging into the service. Check the checkbox next to the Developer Agreement and click the Create your Twitter application button.

Once the app is created, click on the Keys and Access Tokens tab and note the Consumer Key (API Key) and Consumer Secret (API Secret) fields, as shown in **Figure 3**. You'll use them shortly.

Creating the UI

Open the MainPage.xaml file and add the XAML in **Figure 4**. Note that there's an added namespace for the controls in the UWP Community Toolkit. This is where the BladeView control resides:

```
xmlns:controls="using:Microsoft.Toolkit.Uwp.UI.Controls"
```

Introducing the BladeView Control

The BladeView control will look familiar to users of the Azure Portal Web site (portal.azure.com). If you're unfamiliar with it, the BladeView control provides a container to host "blades" or tiles. The XAML in **Figure 4** includes a "DummyBlade" to keep the XAML designer view from crashing. It'll throw an exception if it encounters a BladeView without any BladeItems. Because the IsOpen property is set to False, users will never see the BladeItem.

Logging into Twitter

Next, connect the app to the Twitter API by adding the following event handler for the btnLogin_Click event:

```
private async void btnLogin_Click(object sender, RoutedEventArgs e)
  {
    string apiKey = "pkfAUvqfMAGr53D4huKOzDYDP";
    string apiSecret =
      "bgJCH9ESj1wraCoHBI5OqEqhkac1AOZxujqvnCWKNRJgBMhyPG";
    string callbackUrl = "http://www.franksworld.com";

    TwitterService.Instance.Initialize(apiKey, apiSecret, callbackUrl);

    if (await TwitterService.Instance.LoginAsync())
    {
      splSearch.Visibility = Visibility.Visible;
    }
  }
```

The code uses the API Key, API Secret, and Callback URL fields and uses them in the parameters of the Initialize method of the TwitterService.Instance. TwitterService.Instance is a singleton that will maintain state throughout the entire app. Calling the LoginAsync method initiates the call to the Twitter API. If the login is successful, the method returns true. In that case, you should make the StackPanel with the Search Controls visible.

Displaying Search Results

With the Twitter API calls set up, it's time to create a place for the search results to be displayed. To do this, you'll create a user control. The user control will contain code to perform the Twitter API search, as well as host the necessary controls to display the search results.

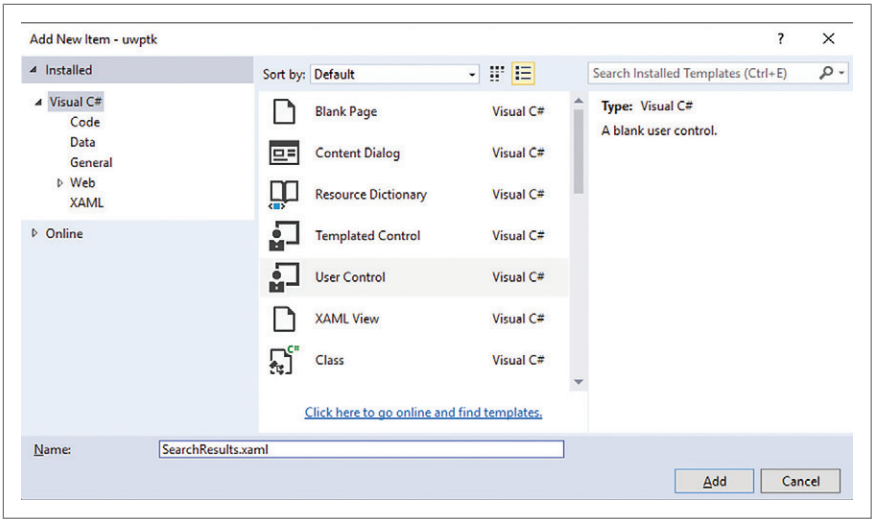


Figure 5 **Adding a New User Control to the Project**

To get started, right-click on the project and choose Add | New Item in the context menu. In the following dialog box, look for user control. Name the user control SearchResults and click Add, as shown in **Figure 5**.

Modify the SearchResults.xaml file to add the XAML found in **Figure 6**.

The XAML contains a ListView and the necessary DataTemplate to display the Twitter search results. Open the SearchResults.xaml.cs file and add the following property to the Search Results class:

```
public string SearchTerm { get; private set; }
```

Then, modify the constructor to add a string parameter for the search term:

```
public SearchResults(string searchTerm)
{
  this.InitializeComponent();
  this.SearchTerm = searchTerm;
  Search();
}
```

Now, add the following method:

```
private async void Search()
  {
    lvSearchResults.ItemsSource = await
    TwitterService.Instance.SearchAsync(this.SearchTerm, 50);
  }
```

The Search method calls the SearchAsync method with two parameters: the search term and the limit of results to return. All the underlying REST API plumbing work is done by the UWP Community Toolkit.

Now that the SearchResults user control is ready, it's time to add code to the MainPage.xaml.cs file to complete the app. Add the following event handler for the btnSearch Button control:

```
private void btnSearch_Click(object sender, RoutedEventArgs e)
  {
    BladeItem bi = new BladeItem();
    bi.Title = txtSearch.Text;
    bi.Content = new SearchResults(txtSearch.Text);
    bladeView.Items.Add(bi);
  }
```

The BladeView control can contain any number of BladeItems. The previous code snippet creates a BladeItem control and sets the Title of the BladeItem to the text from the search textbox. Next, it sets the contents of the BladeItem control to a new instance of the SearchResults user control, passing the search term off to the constructor. Finally, it adds the BladeItem to the BladeView.

Run the solution now. Click the Log In button. When prompted, enter your Twitter credentials and grant the app the permissions it's asking for. The window will close and the search panel will now be visible. After entering a few search terms, your screen should look something like **Figure 7**.

Adding the Copy Function

Now that you have all the tweets you're interested in neatly organized by blade, you need a way to get the data into a text format. Ideally, you'd like to be able to right-click (or tap, if on a touchscreen device) and copy the contents of the tweet to the clipboard. Adding this feature requires some modification to the XAML and code for the SearchResults user control.

Inside the SearchResults.xaml file, you want to add a flyout menu to the ListView control. Inside the ListView tag add the following XAML to create a MenuFlyout as a resource within the ListView control:

```
<ListView.Resources>
  <MenuFlyout x:Name="mfCopyMenu">
    <MenuFlyout.Items>
      <MenuFlyoutItem Name="mfiCopy" Text="Copy" Click="mfiCopy_Click"/>
    </MenuFlyout.Items>
  </MenuFlyout>
</ListView.Resources>
```

> All the underlying REST API plumbing work is done by the UWP Community Toolkit.

While still in the SearchResults.xaml file, add the following event handler to the ListView control to detect when the ListView is right-clicked or tapped:

```
RightTapped="lvSearchResults_RightTapped"
```

Figure 6 **XAML for the SearchResults User Control**

```
<Grid>
  <ListView Name="lvSearchResults" Width="350" >
  <ListView.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal" VerticalAlignment="Top" Margin="0,4,4,0">
        <Image Source="{Binding User.ProfileImageUrl}" Width="64"
          Margin="8"  ></Image>
          <StackPanel Width="240">
            <TextBlock Text="{Binding Text}"
              TextWrapping="WrapWholeWords"></TextBlock>
            <TextBlock Text="{Binding CreationDate}"
              FontStyle="Italic" ></TextBlock>
            <TextBlock Text="{Binding User.ScreenName}"
              Foreground="Blue"></TextBlock>
            <TextBlock Text="{Binding User.Name}"></TextBlock>
          </StackPanel>
        </StackPanel>
      </DataTemplate>
    </ListView.ItemTemplate>
  </ListView>
</Grid>
```

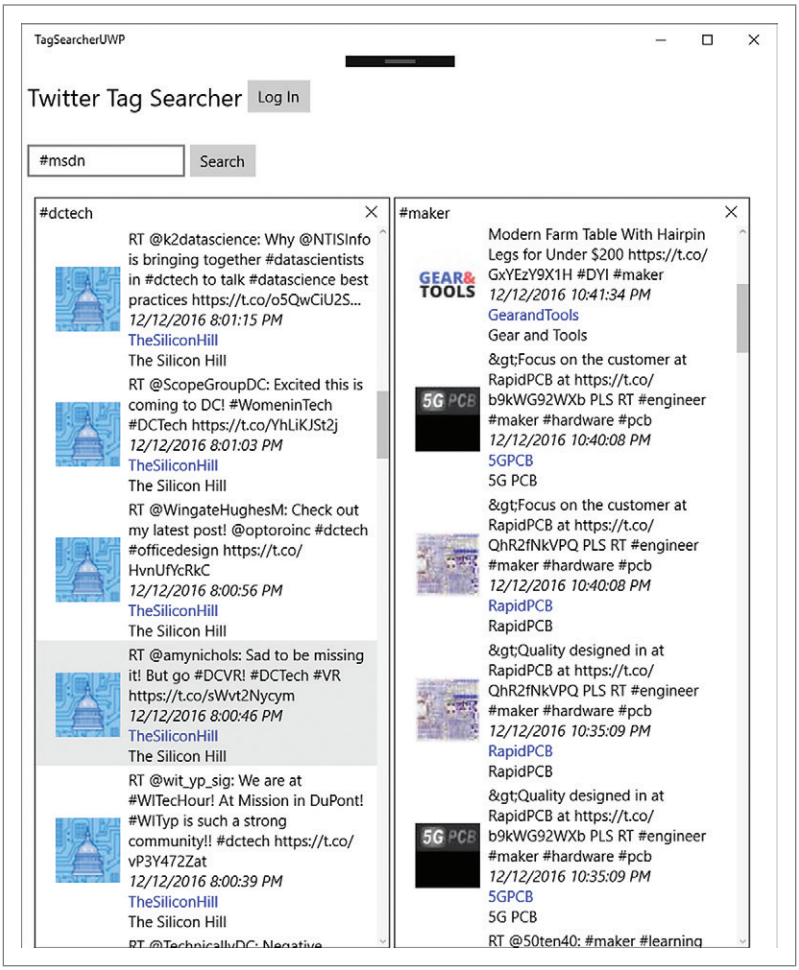


Figure 7 **The Tag Search App in Action**

Now add the following event handler code in the SearchResults.xaml.cs file:

```
private void lvSearchResults_RightTapped(object sender,
RightTappedRoutedEventArgs e)
{
  var tweet = ((FrameworkElement)e.OriginalSource).DataContext;
  mfiCopy.Tag = tweet;
  mfCopyMenu.ShowAt(lvSearchResults, e.GetPosition(lvSearchResults));
}
```

The purpose of this code is to capture the tweet object from the DataContext and store it into the MenuFlyoutItem Tag property. The Tag property is inherited from FrameworkElement and is meant to store custom information about an object. Once the selected tweet object is stored in the Tag property of the MenuFlyoutItem, it's time to display the flyout menu. Users expect a context menu to appear where they clicked or tapped on the screen. That's why the code sends event position information to the ShowAt method.

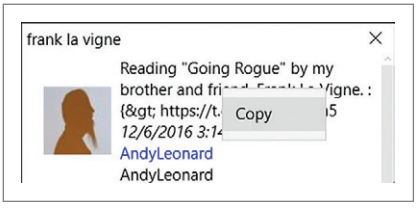


Figure 8 **Testing the Copy Context Menu Function on a Sample Tweet**

Now it's time to add the event handler for the MenuFlyoutItem control and code to copy the contents of the tweet to the clipboard. Add the following event handler to the SearchResults.xaml.cs file:

```
private void mfiCopy_Click(object sender, RoutedEventArgs e)
{
  var menuFlyoutItemSender = (MenuFlyoutItem)sender;
  var tweet = menuFlyoutItemSender.Tag as Tweet;

  DataPackage dataPackage = new DataPackage();
  dataPackage.RequestedOperation = DataPackageOperation.Copy;
  dataPackage.SetText($"@{tweet.User.ScreenName} {tweet.Text} ");
  Clipboard.SetContent(dataPackage);
}
```

The first two lines of code retrieve the tweet data from the Tag property of the MenuFlyoutItem. Once that's obtained, it's time to send data to the clipboard. In UWP apps, this is done by using the DataPackage class. A full exploration of the DataPackage class is beyond the scope of this column; however, if you're interested in learning more, I recommend reading the "Copy and Paste" documentation page at bit.ly/2h54IK0. The "DataPackage Class" documentation page is at bit.ly/2hpo2Fc.

The clipboard can handle robust formatting of text and images. However, for this column, I'm interested only in the text contents of the tweet and the Twitter handle of the person who made it. The UWP Community Toolkit stores that as ScreenName inside the User object. Finally, I set the contents of the Clipboard to the DataPackage object.

Run the solution now, log in, and enter a search term. Find a tweet you wish to copy, right-click or tap to see the context menu, as shownin **Figure 8**. Click Copy.

Now, run Notepad, or your favorite text editor, and chose Edit | Paste or use Ctrl+V. You should see this text from the tweet: @AndyLeonard Reading "Going Rogue" by my brother and friend, Frank La Vigne. :{> https://t.co/JnuAzqO6m5.

Wrapping Up

As you can see, the UWP Community Toolkit facilitates rapid development of cloud-connected UWP apps. It only took one line of code to log into Twitter. Searching Twitter was equally as brief. Most of the code had more to do with the presentation of the data and how users interact with it. The UWP Community Toolkit provides rich UI controls, as well as straightforward ways to access popular cloud APIs such as Twitter. Low-level REST API and authentication mechanisms are abstracted away into a clean IntelliSense-enabled API. This enables developers to focus on how users interact with the data, rather than obtaining the data. The UWP Community Toolkit can make any UWP app better and easier to connect to social media and other cloud services. ■

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

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

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

EVENT PARTNERS

Microsoft

Magenic

PLATINUM SPONSORS

ActivePDF®

Alachisoft®

SUPPORTED BY

Visual Studio

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

Development Topics include:

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

TURN THE PAGE FOR FULL AGENDA DETAILS

Sunday Pre-Con Hands-On Labs

Choose From:

- Angular
- Azure
- XAML

NEW! ONLY $645

SPACE IS LIMITED

Register by February 17 and Save $300!*

Use promo code VSLFEB4

*Available on 3 and 5 day packages only.

REGISTER NOW

ROCK YOUR CODE TOUR 2017

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

msdn magazine | Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA INC. YOUR GROWTH. OUR BUSINESS.

vslive.com/lasvegasmsdn

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

AGENDA AT-A-GLANCE

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

CONNECT WITH VISUAL STUDIO LIVE!

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

REGISTER NOW

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

START TIME	END TIME	
		Full Day Hands-On Labs: Sunday, March 12, 2017 *(Separate entry fee required)*
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries
9:00 AM	1:00 PM	**HOL01** Full Day Hands-On Lab: Build an Azure App in a Day - *Brian Randell*
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas
2:00 PM	6:00 PM	Workshop Continues
		Pre-Conference Workshops: Monday, March 13, 2017 *(Separate entry fee required)*
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries
9:00 AM	1:00 PM	**M01** Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries & Roy Cornelissen*
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas
2:00 PM	6:00 PM	Workshop Continues
7:00 PM	9:00 PM	Dine-A-Round
		Day 1: Tuesday, March 14, 2017
8:00 AM	9:00 AM	**Keynote:** Rub DevOps On It - *Donovan Brown, Senior DevOps Program Manager,*
9:15 AM	10:30 AM	**T01** Essential Web Development with ASP.NET Core - *Mark Michaelis* / **T02** An Overview of the Xamarin Programming Platforms - *Laurent Bugnion*
10:45 AM	12:00 PM	**T06** Migrating to ASP.NET Core - A True Story - *Adam Tuliper* / **T07** Building Truly Universal Applications with Windows, Xamarin and MVVM - *Laurent Bugnion*
12:00 PM	1:00 PM	Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors
1:30 PM	2:45 PM	**T11** An Introduction to TypeScript - *Jason Bock* / **T12** What's New for Developers in SQL Server 2016 - *Leonard Lobel*
3:00 PM	4:15 PM	**T16** Assembling the Web - A Tour of WebAssembly - *Jason Bock* / **T17** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*
4:15 PM	5:30 PM	Welcome Reception
		Day 2: Wednesday, March 15, 2017
8:00 AM	9:15 AM	**W01** Angular 101: Part 1 - *Deborah Kurata* / **W02** Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - *Laurent Bugnion*
9:30 AM	10:45 AM	**W06** Angular 101: Part 2 - *Deborah Kurata* / **W07** Windows for Makers: Raspberry Pi, Arduino & IoT - *Nick Landry*
11:00 AM	12:00 PM	**General Session:** To Be Announced, *James Montemagno, Principal Program*
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)
1:30 PM	2:45 PM	**W11** User Authentication for ASP.NET Core MVC applications - *Brock Allen* / **W12** Cloud Enable an Existing WPF LOB App - *Robert Green*
3:00 PM	4:15 PM	**W16** Securing Web APIs in ASP.NET Core - *Brock Allen* / **W17** Strike Up a Conversation with Cortana on Windows 10 - *Walt Ritscher*
4:30 PM	5:45 PM	**W21** ASP.NET Core 1.0 Tag Helpers - *Robert Boedigheimer* / **W22** Busy Developer's Guide to NoSQL - *Ted Neward*
7:00 PM	8:30 PM	Experience The LINQ Vortex & High Roller Event
		Day 3: Thursday, March 16, 2017
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**TH01** Debugging Your Website with Fiddler and Chrome Developer Tools - *Robert Boedigheimer* / **TH02** Busy .NET Developer's Guide to Native iOS - *Ted Neward*
9:30 AM	10:45 AM	**TH06** I Say A "Front-end Build Pipeline", You Say WAT!? - *Chris Klug* / **TH07** Building Cross-Platform Business Apps with CSLA .NET - *Rockford Lhotka*
11:00 AM	12:15 PM	**TH11** JavaScript Patterns for the C# Developer - *Ben Hoelting* / **TH12** Building Connected and Disconnected Mobile Apps - *James Montemagno*
12:15 PM	1:45 PM	Lunch
1:45 PM	3:00 PM	**TH16** Integrating AngularJS & ASP.NET MVC - *Miguel Castro* / **TH17** Native iOS and Android Development with C# and Xamarin - *James Montemagno*
3:15 PM	4:30 PM	**TH21** Increase Website Performance and Search with Lucene.Net Indexing - *Ben Hoelting* / **TH22** Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - *Nick Landry*
		Post-Conference Workshops: Friday, March 17, 2017 *(Separate entry fee required)*
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries
8:00 AM	5:00 PM	**F01** Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*

Speakers and sessions subject to change

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with Magenic

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

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

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

Pre-Conference Workshop Registration - Coffee and Morning Pastries

HOL02 Full Day Hands-On Lab: AngularJS 2 - *Ted Neward*	**HOL03** Full Day Hands-On Lab: An Introduction to Building XAML Applications - *Billy Hollis*

Lunch @ Le Village Buffet, Paris Las Vegas

Workshop Continues	Workshop Continues

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

Pre-Conference Workshop Registration - Coffee and Morning Pastries

M02 Workshop: Developer Dive into SQL Server 2016 - *Leonard Lobel*	**M03** Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka & Jason Bock*	**M04** Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*

Lunch @ Le Village Buffet, Paris Las Vegas

Workshop Continues	Workshop Continues	Workshop Continues

Dine-A-Round

Day 1: Tuesday, March 14, 2017

US Developer Division Team, Microsoft

T03 What's New in Visual Studio 2017 - *Robert Green*	**T04** Understanding the VR/AR Landscape - *Katherine Harris*	**T05** Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*
T08 Roll Your Own Dashboard in XAML - *Billy Hollis*	**T08** Mobile DevOps with the Microsoft Stack - *Abel Wang*	**T10** Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*

Lunch

Dessert Break - Visit Exhibitors

T13 A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*	**T14** Make PDF Work For You - *Aaron Schnarr*	**T15** Architecture: The Key to Modern App Success - *Brent Edwards*
T18 Essential C# 7.0 - *Mark Michaelis*	**T19** To Be Announced	**T20** Focus on the User Experience #FTW - *Jim Barrett*

Welcome Reception

Day 2: Wednesday, March 15, 2017

W03 What's New in Azure IaaS v2 - *Eric D. Boyd*	**W04** Tactical DevOps with VSTS - *Brian Randell*	**W05** DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*
W08 Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W09** Use Visual Studio to Scale Agile in Your Enterprise - *Richard Hundhausen*	**W10** Mobile Panel - *James Montemagno, Ryan J. Salva, Kevin Ford, Rockford Lhotka*

Manager, Xamarin Microsoft

Birds-of-a-Feather Lunch

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

W13 I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - *Adam Tuliper*	**W14** Professional Scrum Development Using Visual Studio 2017 - *Richard Hundhausen*	**W15** C# Everywhere: How CSLA .NET Enables Amazing Cross-Platform Code Reuse - *Rockford Lhotka*
W18 Cloud Oriented Programming - *Vishwas Lele*	**W19** Introduction to Containers and Docker - *Marcel de Vries*	**W20** Coding for Quality and Maintainability - *Jason Bock*
W23 Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*	**W24** Using Docker on Windows in VSTS Build and Release Management - *Marcel de Vries*	**W25** Modern Mobile Development: Build a Single App For iOS & Android with Xamarin Forms - *Kevin Ford*

Experience The LINQ Vortex & High Roller Event

Day 3: Thursday, March 16, 2017

Registration - Coffee and Morning Pastries

TH03 Accelerate Your Mobile App Development with Azure App Services Mobile Apps - *Brian Noyes*	**TH04** Agile: You Keep Using That Word - *Philip Japikse*	**TH05** Modern Web Development: Building Server Side Using ASP.NET Core, MVC, Web API, and Azure - *Allen Conway*
H08 Connect All The Things with Azure Service Bus, Notification Hubs, Event Hubs, and IoT Hubs - *Brain Noyes*	**TH09** Visualizing the Backlog with User Story Mapping - *Philip Japikse*	**TH10** Modern Web Development: Building Client Side Using TypeScript and Angular - *Allen Conway*
TH13 Add A Conversational User Interface to Your App with the Microsoft Bot Framework - *Walt Ritscher*	**TH14** End-to-End Dependency Injection & Testable Code - *Miguel Castro*	**TH15** Cloud Panel - *Rockford Lhotka*

Lunch

TH18 Introduction to R and Microsoft R Server - *James McCaffrey*	**TH19** Open Source Software for Microsoft Developers - *Rockford Lhotka*	**TH20** Universal Windows Development: UWP for PC, Tablet & Phone - *Nick Landry*
TH23 Introduction to Azure Machine Learning - *James McCaffrey*	**TH24** SOLID - The Five Commandments of Good Software - *Chris Klug*	**TH25** Using All That Data: Power BI to the Rescue - *Scott Diehl*

Post-Conference Workshops: Friday, March 17, 2017 *(Separate entry fee required)*

Post-Conference Workshop Registration - Coffee and Morning Pastries

F02 Workshop: Practical ASP.NET DevOps with VSTS or TFS - *Brian Randell*	**F03** Workshop: Creating Experiences for the HoloLens with Unity - *Nick Landry & Adam Tuliper*	**F04** Workshop: Modern App Deep Dive: Xamarin, Responsive Web, UWP, CSLA .NET - *Kevin Ford, Jason Bock, Brent Edwards, Allen Conway*

vslive.com/lasvegasmsdn

Backlash

Thou shalt not make a machine in the likeness of a man's mind.
—From the "Orange Catholic Bible," in "Dune," by Frank Herbert (Chilton, 1965)
Thou and I are first-class sinners then, aren't we?
—From "Why Software STILL Sucks," by David S. Platt (to be published in 2018)

Anniversaries are natural times for reflection. As I begin my eighth year fulminating in this space, I see today's software industry crossing a watershed.

Software, up until now, has meant machines aiding humans in simple tasks. My favorite example is the auto-complete in Microsoft Word. When I type "hte," instead of ringing an alarm bell and insisting that I fix it, Word simply corrects it to the "the" that I intended. I praised this behavior in my very first Don't Get Me Started column in February 2010 (msdn.com/magazine/ee309884). "This feature uses the computer to do what computers do best, so that the human user can do what humans do best. It understands; it respects; it even *enhances* the humanity of the user." Great.

> Our software, and therefore we as its creators, is shouldering much more responsibility today than it ever has. What happens if we get it wrong?

Today's artificial intelligences do not merely run code written by their developers ("case WM_LBUTTONDOWN:" and so on). Today's machines are starting to evolve their own behavior via processing large data sets, modifying—and we hope improving—themselves in ways not directly controlled by their human creators. Perhaps these evolutions will be beneficial, or at least benign. I certainly go to work every day aiming for those results, and I know you do, too. But today we're entering a realm of critical mass, where we plug stuff in, but we're not quite sure what's going to come out.

This isn't just Clippy popping up, emitting his little mechanical fart and saying: "I see you're writing a crude forgery. Can I help? Is this a business forgery or a personal forgery?" We're handing over life-and-death responsibility to these new AIs.

Consider self-driving cars, now being tested on the open road and coming soon to a street near you. If the computer driving the car is boxed in and can't avoid hitting something, does it choose the bicyclist wearing a helmet, because he'll probably be hurt less? Or the one not wearing a helmet, because he's obviously less intelligent? Does it matter if either is flouting the traffic signal? How about if the car's passenger isn't wearing a seat belt? You can game this out in the Moral Machine simulator at moralmachine.mit.edu.

Our software, and therefore we as its creators, is shouldering much more responsibility today than it ever has. What happens if we get it wrong?

In his 1965 masterwork "Dune," Frank Herbert envisions a far-future universe in which interstellar travel is commonplace, but which doesn't have any sort of computing machinery. The machines had gotten too smart, and humans had to band together to wipe them out in the Butlerian Jihad, set 10,000 years previously. The surviving humans then forbade the resurgence of computing machinery with the biblical commandment you see at the start of this column.

"Humanity lost its drive," reads the prequel ("Butlerian Jihad" by Herbert's son Brian Herbert, Tor Books, 2010). "With few ambitions, most people allowed efficient machines to perform everyday tasks for them. Gradually humans ceased to think, or dream … or truly live." Is this where our Nest thermostats and, soon, our self-driving Uber cars are leading us?

As Reverend Mother Gaius Helen Mohiam instructs the young Paul Atreides: "Once men turned their thinking over to machines in the hope that this would set them free. But that only permitted other men with machines to enslave them." Does the alleged hack of the 2016 Democratic campaign ring any bells here?

If you and I can't make the world work, the alternative isn't pretty. In his acclaimed novel, "A Canticle for Leibowitz" (Lippincott, 1960), Walter E. Miller Jr. describes a post-apocalyptic purge of brainy people. "Nothing had been so hateful in the sight of those mobs as the man of learning … Joyfully the mobs accepted the name, took up the cry: *Simpletons! Yes, yes! I'm a simpleton. … Anybody here not a simpleton? Get the bastard, if there is!*"

We'd better get this new stuff right. Because if we get it wrong, they're coming after us. ■

DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

Looking for a powerful distributed cache?

Get the easiest, most powerful in-memory data grid designed to scale your .NET applications.

- Fast and linearly scalable
- Enterprise-grade availability
- Industry-leading ease of use
- Integrated in-memory computing

Go with the industry leader in distributed caching

Trusted by hundreds of enterprise customers for more than 12 years, ScaleOut's distributed caching technology delivers rock-solid performance, legendary ease of use, and world-class support. Unlike Redis, it offers a better migration path from AppFabric Caching. Here's why:

	ScaleOut	Redis
Source-code compatible AppFabric Caching APIs for seamless migration	✔	✖
Architected from the ground up for automatic scaling and high availability	✔	✖
Fully coherent data storage and access for mission-critical applications	✔	✖
Advanced .NET features: distributed LINQ query, C# MapReduce, and more	✔	✖

Replacing AppFabric Caching? Trouble switching to Redis?
Check out our AppFabric-compatible drop-in: **www.scaleoutsoftware.com/appfabric**

ScaleOut Software

Please visit **scaleoutsoftware.com** or call us at **+1.503.643.3422** to learn how ScaleOut's technology can help accelerate your business.