THE MICROSOFT JOURNAL FOR DEVELOPERS

APRIL 2018 VOL 33 NO 4

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
and experience the DevExpress difference today.

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

APRIL 2018 VOL 33 NO 4

msdn magazine

COLUMNS

Microsoft

INFRAGISTICS

Infragistics Ultimate 17.2

Productivity Tools & Fast Performing UI Controls for Quickly Building Web, Desktop, & Mobile Apps

Includes 100+ beautifully styled, high-performance grids, charts & other UI controls, plus visual configuration tooling, rapid prototyping, and usability testing.

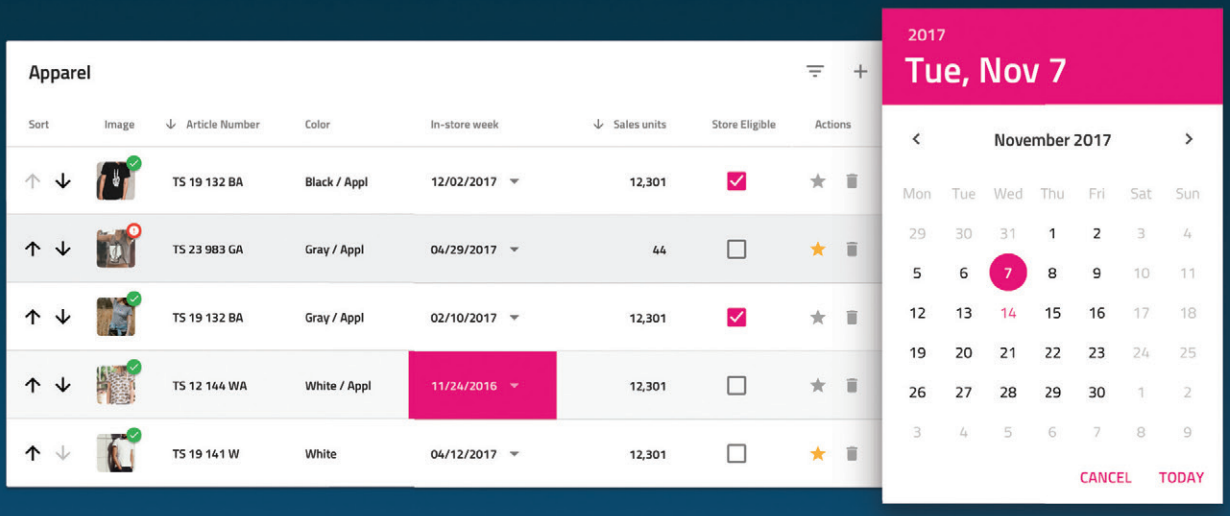
Angular | JavaScript / HTML5 | ASP.NET | Windows Forms | WPF | Xamarin

Download a free trial at
Infragistics.com/Ultimate

Featuring

Ignite UI

A complete UI component library for building high-performance, data rich web applications



- Create beautiful, touch-first, responsive desktop & mobile wep apps with over 100 JavaScript / HTML5, MVC & **Angular components**.
- Our easy to use Angular components have no 3rd party dependencies, a tiny footprint, and easy-to-use API.
- The Ignite UI **Angular Data Grid** enables you to quickly bind data with little coding - including features like sorting, filtering, paging, movable columns, templates and more!
- Speed up development time with responsive layout, powerful data binding, cross-browser compatibility, WYSIWYG page design, & built-in-themes.

Download a free trial of Ignite UI at: **Infragistics.com/Ignite-ui**

To speak with our sales team or request a product demo call: 1.800.321.8588

msdn magazine

APRIL 2018 VOLUME 33 NUMBER 4

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 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
Web: 1105Reprints.com

LIST RENTAL This publication's subscriber list is not available for rental. However, other lists from 1105 Media, Inc. can be rented.
For more information, please contact our list manager:
Jane Long, Merit Direct
Phone: 913-685-1301;
E-mail: jlong@meritdirect.com;
Web: meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail.
E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618
Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311
The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

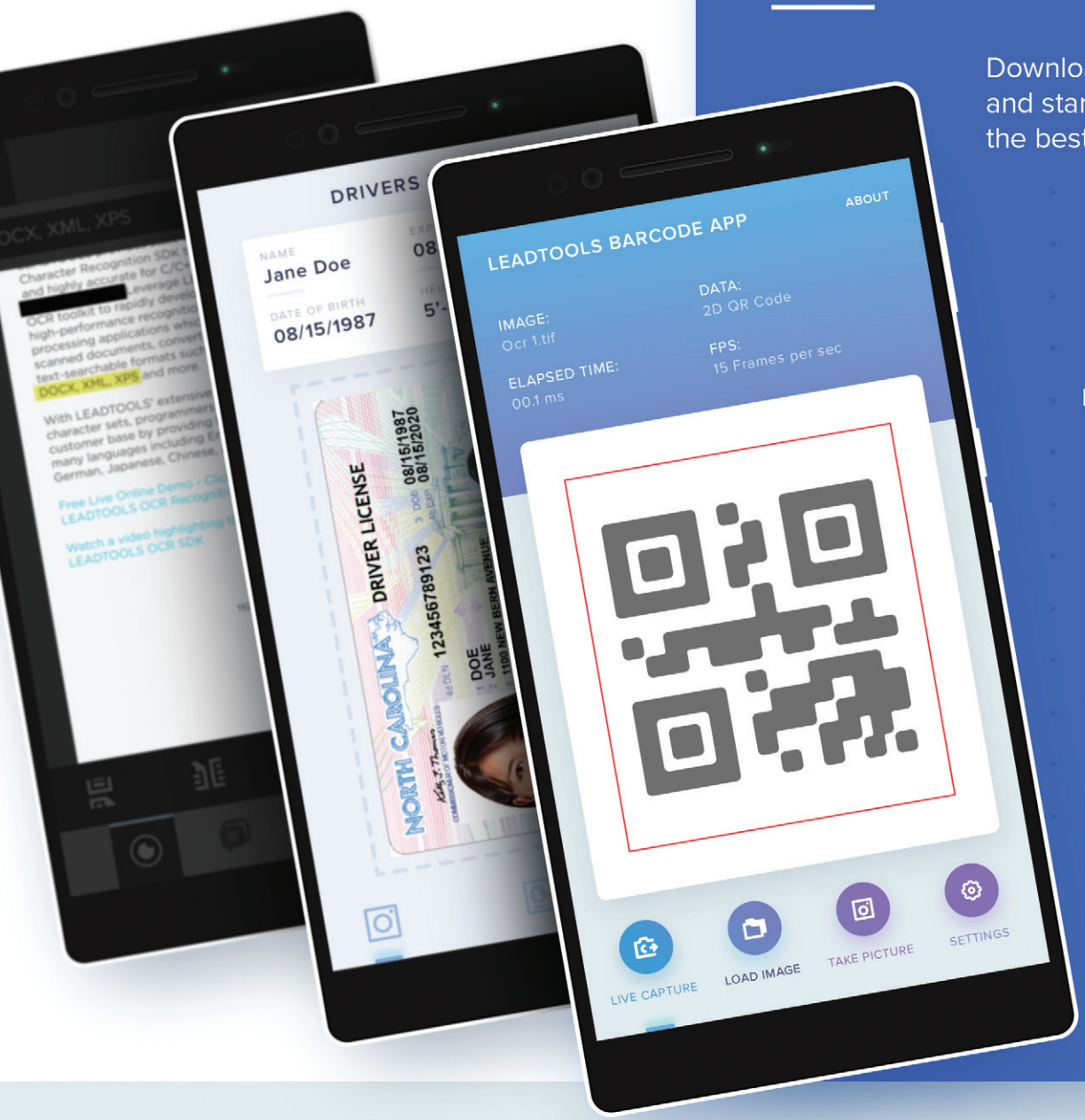
BPA WORLDWIDE
1906 2006 100 AMERICAN BUSINESS MEDIA
WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

WORLD-CLASS MOBILE RECOGNITION SDK

Quickly and accurately extract data from any document or image with LEADTOOLS. Developers using the SDK easily add advanced OCR, Barcode, Forms, Driver's License, Passport, Check, and Credit Card recognition functionality into their applications.



Download the SDK today and start developing with the best in:

- RELIABILITY
- ACCURACY
- SPEED

Fully-functional sample apps built with LEADTOOLS are also available for download from **Google Play**, **App Store**, and **Microsoft Store**.

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Editor's Note

MICHAEL DESMOND

100

One. Hundred. That's the number of columns David Platt has written for the back page of *MSDN Magazine* since he first came on board back in February 2010. And appropriately enough, the 100-column milestone lands in this month's April issue, coinciding with April Fools' Day.

David is no stranger to humor (though his humor can get a little strange). His April 2011 column, "The Cat Butt Factor" (msdn.com/magazine/gg983482) remains one of my all-time favorites. Guest-written by his then-20-year-old cat Simba, the column featured gems of wisdom amid the kitty litter, including this summation of his owner's technical prowess:

"He's supposed to know this stuff. He's written more than a dozen books on programming Windows, he teaches it at a major university and Microsoft designated him a Software Legend back in 2002. And a cat's butt has made him repave his hard drive, more than once. He thinks it's wrong. I think it shows how we cats really run the world."

There's also the challenge that Simba posed to developers to write programs that can weather unpredictable events—like cats lounging all over PC keyboards: "Legend says that the Internet was designed to survive a nuclear war. Your programs ought to survive the attack of a cat's butt."

True wisdom, there. Over the years, David has written again and again about difficult UIs and the terrible costs they impose, including "This Is Not a Drill" (msdn.com/magazine/mt845656), "A Measure of Displeasure" (msdn.com/magazine/mt826357) and his very first column, "The Human Touch" (msdn.com/magazine/ee309884). He's taken aim at the computer-deficient medical field, the tectonic forces reshaping education, and the high-risk lifestyle of selfie-obsessed smartphone users.

You might be tempted to think that David's column is a parade of rants, jokes and complaints, but you'd be wrong. Columns like "Jose, Can You C?" (msdn.com/magazine/dn574807) and "Imagine That" (msdn.com/magazine/hh456410) celebrate the achievement and drive of those new to our shores. He's honored the accomplishments of giants like Grace Hopper, Steve Jobs and Dennis Ritchie, and provided a glimpse into the amazing efforts of young student programmers creating a mobile app for physicians treating patients with congestive heart failure.

> Thank you, David, for 100 months of unpredictable wisdom. And be sure to pass on our thanks to Simba, too.

It can be a tricky thing, putting an outspoken author on the back page of a company-owned magazine, especially one who writes repeatedly about his love of "pouring oil on troubled fires." And yet here we are, 100 columns later, celebrating the slow-motion balancing act that's brought David's unique wit and wisdom in equal measure to *MSDN Magazine*.

It brings to mind a quote from one of my favorite Don't Get Me Started columns, "Left Brains for the Right Stuff" (msdn.com/magazine/mt703442). In it, David celebrates the scientists, programmers and mathematicians who put humanity on the moon, and recounts a line he overheard from one of the scientists talking about the effort at the time:

"We didn't know what we couldn't do. So we just went ahead and did it."

I think that just about sums up the experience for all of us, present company included. Thank you, David, for 100 months of unpredictable wisdom. And be sure to pass on our thanks to Simba, too.

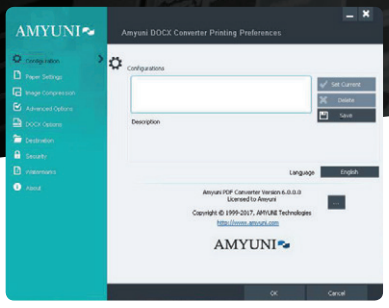
Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

DOCXCONVERTER
For Windows

Free Demo at DOCXConverter.com

Amyuni DOCX Converter for Windows

Convert any document, including PDF documents, into DOCX format.
Enable editing of documents using Microsoft Word or other Office products.



A standalone desktop version, a server product for automated processing or an SDK for integration into third party applications.

Create

Create naturally editable DOCX documents with paragraph formatting and reflow of text

Convert

Convert images and graphics of multiple formats into DOCX shapes

OCR

Use OCR technology to convert non-editable text into real text

Extract

Extract headers and footers from source document and save them as DOCX headers and footers

Open

Open PDF documents with the integrated PDF viewer and quickly resave them to DOCX format

Configure

Configure the way the fonts are embedded into the DOCX file for optimal formatting

A virtual printer driver available for Windows 7 to Windows 10 and Windows Server 2008 to 2016

Powered by Amyuni Technologies:

Developers of the Amyuni PDF Converter and Amyuni PDF Creator products integrated into hundreds of applications and installed on millions of desktops and servers worldwide.

www.docxconverter.com

JULIE LERMAN

EF Core 2 Owned Entities and Temporary Work-Arounds

The new Owned Entity feature in EF Core 2.0 replaces the Complex Type feature of Entity Framework "classic" (EF thru EF6). Owned Entities allow the mapping of value objects to the data store. It's quite common to have a business rule that permits properties based on value objects to be null. Moreover, because value objects are immutable, it's also important to be able to replace properties that contain a value object. The current version of EF Core doesn't allow either of these scenarios by default, although both will be supported in upcoming iterations. In the meantime, rather than treating these limitations as showstoppers for those of us who love the benefits of domain-driven design (DDD) patterns, this article will show you how to work around those limitations. A DDD practitioner may still reject these temporary patterns as not following the principles of DDD closely enough, but the pragmatist in me is satisfied, buttressed by the knowledge that they are simply temporary solutions.

Notice that I said "by default." It turns out that there is a way to have EF Core take responsibility to enforce its own rule about null owned entities and allow for value object replacement, without dramatically affecting your domain classes or your business rules. In this column, I'll show you how to do this.

> Given that a value object is composed of its properties and then, as a whole, used as a property in another class, persisting its data takes some special effort.

There is another way around the nullability problem, which is to simply map the value object type to its own table, thus physically splitting the value object data from the rest of the object to which it belongs. While this may be a good solution for some scenarios, it's generally one I don't want to use. Therefore, I prefer to use my work-around, which is the focus of this column. But first I want to ensure you understand why this problem and solution are important enough that I'm devoting this column to the topic.

Code download available at msdn.com/magazine/0418magcode.

A Short Primer on Value Objects

Value objects are a type that lets you encapsulate multiple values into a single property. A string is a great example of a value object. Strings are made up of a collection of chars. And they are immutable—immutability is a critical facet of a value object. The combination and order of the letters c, a and r have a specific meaning. If you were to mutate it, for example by changing the last letter to a "t," you'd completely change the meaning. The object is defined by the combination of all of its values. As such, the fact that the object can't be modified is part of its contract. And there's another important aspect of a value object—it doesn't have its own identity. It can be used only as a property of another class, just like a string. Value objects have other contractual rules, but these are the most important ones to start with if you're new to the concept.

Given that a value object is composed of its properties and then, as a whole, used as a property in another class, persisting its data takes some special effort. With a non-relational database such as a document database, it's easy to just store the graph of an object and its embedded value objects. But that's not the case when storing into a relational database. Starting with the very first version, Entity Framework included the ComplexType, which knew how to map the properties of the property to the database where EF was persisting your data. A common value object example is PersonName, which might consist of a FirstName property and a LastName property. If you have a Contact type with a PersonName property, by default, EF Core will store the FirstName and LastName values as additional columns in the table to which Contact is mapped.

An Example of a Value Object in Use

I've found that looking at a variety of examples of value objects helped me to better understand the concept, so, I'll use yet another example—a SalesOrder entity and a PostalAddress value object. An order typically includes both a shipping address and a billing address. While those addresses may exist for other purposes, within the context of the order, they're an integral part of its definition. If a person moves to a new location, you still want to know where that order was shipped, so it makes sense to embed the addresses into the order. But in order to treat addresses consistently in my system, I prefer to encapsulate the values that make up an address in their own class, PostalAddress, as shown in **Figure 1**.

Figure 1 **PostalAddress ValueObject**

```
public class PostalAddress : ValueObject<PostalAddress>
{
  public static PostalAddress Create (string street, string city,
                                      string region, string postalCode)
  {
    return new PostalAddress (street, city, region, postalCode);
  }
  private PostalAddress () { }
  private PostalAddress (string street, string city, string region,
                         string postalCode)
  {
    Street = street;
    City = city;
    Region = region;
    PostalCode = postalCode;
  }
  public string Street { get; private set; }
  public string City { get; private set; }
  public string Region { get; private set; }
  public string PostalCode { get; private set; }
  public PostalAddress CopyOf ()
  {
    return new PostalAddress (Street, City, Region, PostalCode);
  }
}
```

PostalAddress inherits from a ValueObject base class created by Jimmy Bogard (bit.ly/2EpKydG). ValueObject provides some of the obligatory logic required of a value object. For example, it has an override of Object.Equals, which ensures that every property is compared. Keep in mind that it makes heavy use of reflection, which may impact performance in a production app.

Two other important features of my PostalAddress value object are that it has no identity key property and that its constructor forces the invariant rule that every property must be populated. However, for an owned entity to be able to map a type defined as a value object, the only rule is that it have no identity key of its own. An owned entity is not concerned with the other attributes of a value object.

Figure 2 **The SalesOrder Class Contains Properties That Are PostalAddress Types**

```
public class SalesOrder
{
  public SalesOrder (DateTime orderDate, decimal orderTotal)
  {
    OrderDate = orderDate;

    OrderTotal = orderTotal;
    Id = Guid.NewGuid ();
  }
  private SalesOrder () { }
  public Guid Id { get; private set; }
  public DateTime OrderDate { get; private set; }
  public decimal OrderTotal { get; private set; }
  private PostalAddress _shippingAddress;
  public PostalAddress ShippingAddress => _shippingAddress;

  public void SetShippingAddress (PostalAddress shipping)
  {
    _shippingAddress = shipping;
  }
  private PostalAddress _billingAddress;
  public PostalAddress BillingAddress => _billingAddress;
  public void CopyShippingAddressToBillingAddress ()
  {
    _billingAddress = _shippingAddress?.CopyOf ();
  }
  public void SetBillingAddress (PostalAddress billing)
  {
    _billingAddress = billing;
  }
}
```

Figure 3 **The Migration for the SalesOrder Table, Including All of the Columns for PostalAddress Properties**

```
migrationBuilder.CreateTable(
  name: "SalesOrders",
  columns: table => new
  {
    Id = table.Column(nullable: false)
              .Annotation("Sqlite:Autoincrement", true),
    OrderDate = table.Column(nullable: false),
    OrderTotal = table.Column(nullable: false),
    BillingAddress_City = table.Column(nullable: true),
    BillingAddress_PostalCode = table.Column(nullable: true),
    BillingAddress_Region = table.Column(nullable: true),
    BillingAddress_Street = table.Column(nullable: true),
    ShippingAddress_City = table.Column(nullable: true),
    ShippingAddress_PostalCode = table.Column(nullable: true),
    ShippingAddress_Region = table.Column(nullable: true),
    ShippingAddress_Street = table.Column(nullable: true)
  }
```

With PostalAddress defined, I can now use it as the ShippingAddress and BillingAddress properties of my SalesOrder class (see **Figure 2**). They aren't navigation properties to related data, but just more properties similar to the scalar Notes and OrderDate.

> Owned Entities allow the mapping of value objects to the data store.

These addresses now live within the SalesOrder and can provide accurate information regardless of the current address of the person who placed the order. I will always know where that order went.

Mapping a Value Object as an EF Core Owned Entity

In earlier versions, EF could automatically recognize classes that should be mapped using a ComplexType by discovering that the class was used as a property of another entity and it had no key property. EF Core, however, can't automatically infer owned entities. You must specify this in the DbContext Fluent API mappings in the OnModelCreating method using the new OwnsOne method to specify which property of that entity is the owned entity:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<SalesOrder>().OwnsOne(s=>s.BillingAddress);
  modelBuilder.Entity<SalesOrder>().OwnsOne(s=>s.ShippingAddress);
}
```

I've used EF Core migrations to create a migration file describing the database to which my model maps. **Figure 3** shows the section of the migration that represents the SalesOrder table. You can see that EF Core understood that the properties of PostalAddress for each of the two addresses are part of the SalesOrder. The column names are as per EF Core convention, although you can affect those with the Fluent API.

Additionally, as mentioned earlier, putting the addresses into the SalesOrder table is by convention, and my preference. This

alternate code will split them out to separate tables and avoid the nullability problem completely:

```
modelBuilder.Entity<SalesOrder> ().OwnsOne (
  s => s.BillingAddress).ToTable("BillingAddresses");
modelBuilder.Entity<SalesOrder> ().OwnsOne (
  s => s.ShippingAddress).ToTable("ShippingAddresses");
```

Creating a SalesOrder in Code

Inserting a sales order with both the billing address and shipping address is simple:

```
private static void InsertNewOrder()
{
  var order=new SalesOrder{OrderDate=DateTime.Today, OrderTotal=100.00M};
  order.SetShippingAddress (PostalAddress.Create (
    "One Main", "Burlington", "VT", "05000"));
  order.SetBillingAddress (PostalAddress.Create (
    "Two Main", "Burlington", "VT", "05000"));
  using(var context=new OrderContext()){
    context.SalesOrders.Add(order);
    context.SaveChanges();
  }
}
```

But let's say that my business rules allow an order to be stored even if the shipping and billing address haven't yet been entered, and a user can complete the order at another time. I'll comment out the code that fills the BillingAddress property:

```
// order.BillingAddress=new Address("Two Main","Burlington", "VT", "05000");
```

When SaveChanges is called, EF Core tries to figure out what the properties of the BillingAddress are so that it can push them into the SalesOrder table. But it fails in this case because BillingAddress is null. Internally, EF Core has a rule that a conventionally mapped owned type property can't be null.

EF Core is assuming that the owned type is available so that its properties can be read. Developers may see this as a showstopper for being able to use value objects or, worse, for being able to use EF Core, because of how critical value objects are to their software design. That was how I felt at first, but I was able to create a work-around.

Temporary Work-Around to Allow Null Value Objects

The goal of the work-around is to ensure that EF Core will receive a ShippingAddress, BillingAddress or other owned type, whether or not the user supplied one. That means the user isn't forced to supply a shipping or billing address just to satisfy the persistence layer. If the user doesn't supply one, then a PostalAddress object with null values in its properties will be added in by the DbContext when it's time to save a SalesOrder.

I made a minor adaptation to the PostalAddress class by adding a second factory method, Empty, to let the DbContext easily create an empty PostalAddress:

```
public static PostalAddress Empty()
{
  return new PostalAddress(null,null,null,null);
}
```

Additionally, I enhanced the ValueObject base class with a new method, IsEmpty, shown in **Figure 4**, to allow code to easily determine if an object has all null values in its properties. IsEmpty leverages code that already exists in the ValueObject class. It iterates through the properties and if any one of them has a value, it returns false, indicating that the object is not empty; otherwise, it returns true.

> The goal of the work-around is to ensure that EF Core will receive a PostalAddress whether or not the user supplied one.

But my solution for allowing null owned entities wasn't yet complete. I still needed to use all of this new logic to ensure that new SalesOrders would always have a ShippingAddress and a BillingAddress in order for EF Core to be able to store them into the database. When I initially added this last piece of my solution, I wasn't happy with it because that last bit of code (which I won't bother sharing) was making the SalesOrder class enforce EF Core's rule—the bane of domain-driven design.

Voila! An Elegant Solution

Luckily, I was speaking at DevIntersection, as I do every fall, where Diego Vega and Andrew Peters from the EF team were also presenting. I showed them my work-around and explained what was bothering me—the need to enforce non-null ShippingAddress and BillingAddress in the SalesOrder—and they agreed. Andrew quickly came up with a way to use the work I had done in the ValueObject

Figure 4 **The IsEmpty Method Added to the ValueObject Base Class**

```
public bool IsEmpty ()
{
  Type t = GetType ();
  FieldInfo[] fields = t.GetFields
    (BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public);
  foreach (FieldInfo field in fields)
  {
    object value = field.GetValue (this);
    if (value != null)
    {
      return false;
    }
  }
  return true;
}
```

Figure 5 **Overriding SaveChanges to Provide Values to Null Owned Types**

```
public override int SaveChanges()
{
  foreach (var entry in ChangeTracker.Entries()
            .Where(e => e.Entity is SalesOrder && e.State == EntityState.Added))
  {
    if (entry.Entity is SalesOrder)
    {
      if (entry.Reference("ShippingAddress").CurrentValue == null)
      {
        entry.Reference("ShippingAddress").CurrentValue = PostalAddress.Empty();
      }
      if (entry.Reference("BillingAddress").CurrentValue == null)
      {
        entry.Reference("BillingAddress").CurrentValue = PostalAddress.Empty();
      }
  }
  return base.SaveChanges();
}
```

DevExpress®

DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

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

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

Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

base class and the tweak I made to the PostalAddress to force EF Core to take care of the problem without putting the onus on SalesOrder. The magic happens in the override of the SaveChanges method of my DbContext class, shown in **Figure 5**.

From the collection of entries that the DbContext is tracking, SaveChanges will iterate through those that are SalesOrders flagged to be added to the database, and will make sure they get populated as their empty counterparts.

What About Querying Those Empty Owned Types?

Having satisfied the need for EF Core to store null value objects, it's now time to query them back from the database. But EF Core resolves those properties in their empty state. Any ShippingAddress or BillingAddress that was originally null comes back as an instance with null values in its properties. After any query, I need my logic to replace any empty PostalAddress properties with null.

I spent quite some time looking for an elegant way to achieve this. Unfortunately, there isn't yet a lifecycle hook to modify objects as they're being materialized from query results. There's a replaceable service in the query pipeline called CreateReadValueExpression in the internal EntityMaterializerSource class, but that can only be used on scalar values, not objects. I tried numerous other approaches that were more and more complicated, and finally had a long talk with myself about the fact that this is a *temporary* workaround, so I can accept a simpler solution even if it is has a little bit of code smell. And this task isn't too difficult to control if your queries are encapsulated in a class dedicated to making EF Core calls to the database.

I named the method FixOptionalValueObjects:

```
private static void FixOptionalValueObjects (SalesOrder order) {
  if (order.ShippingAddress.IsEmpty ()) { order.SetShippingAddress (null); }
  if (order.BillingAddress.IsEmpty ()) { order.SetBillingAddress (null); }
}
```

Now I have a solution in which the user can leave value objects null and let EF Core store and retrieve them as non-nulls, yet return them to my code base as nulls anyway.

Replacing Value Objects

I mentioned another limitation in the current version of EF Core 2, which is the inability to replace owned entities. Value objects are by definition immutable. So, if you need to change one, the only way is to replace it. Logically this means that you're modifying the SalesOrder, just as if you had changed its OrderDate property. But because of the way that EF Core tracks the owned entities, it will always think the replacement is added, even if its host, SalesOrder, for example, is not new.

Figure 6 **Making SaveChanges Comprehend Replaced Owned Types by Marking Them as Modified**

```
public override int SaveChanges () {
  foreach (var entry in ChangeTracker.Entries ().Where (
    e => e.Entity is SalesOrder &&
    (e.State == EntityState.Added || e.State == EntityState.Modified))) {
    if (entry.Entity is SalesOrder order) {
      if (entry.Reference ("ShippingAddress").CurrentValue == null) {
        entry.Reference ("ShippingAddress").CurrentValue = PostalAddress.Empty ();
      }
      if (entry.Reference ("BillingAddress").CurrentValue == null) {
        entry.Reference ("BillingAddress").CurrentValue = PostalAddress.Empty ();
      }
      entry.Reference ("ShippingAddress").TargetEntry.State = entry.State;
      entry.Reference ("BillingAddress").TargetEntry.State = entry.State;
    }
  }
  return base.SaveChanges ();
}
```

I made a change to the SaveChanges override to fix this problem (see **Figure 6**). The override now filters for SalesOrders that are added or modified, and with the two new lines of code that modify the state of the reference properties, ensures that ShippingAddress and BillingAddress have the same state as the order—which will either be Added or Modified. Now modified SalesOrder objects will also now be able to include the values of the ShippingAddress and BillingAddress properties in their UPDATE commands.

> Value objects are by definition immutable. So, if you need to change one, the only way is to replace it.

This pattern works because I'm saving with a different instance of OrderContext than I queried, which therefore doesn't have any preconceived notion of the state of the PostalAddress objects. You can find an alternate pattern for tracked objects in the comments of the GitHub issue at bit.ly/2sxMECT.

Pragmatic Solution for the Short-Term

If changes to allow optional owned entities and replacing owned entities were not on the horizon, I'd most likely take steps to create a separate data model to handle the data persistence in my software. But this temporary solution saves me that extra effort and investment and I know that soon I can remove my work-arounds and easily map my domain models directly to my database letting EF Core define the data model. I was happy to invest the time, effort and thought into coming up with the work-arounds so I can use value objects and EF Core 2 when designing my solutions—and help others be able to do the same.

Note that the download that accompanies this article is housed in a console app to test out the solution as well as persist the data to a SQLite database. I'm using the database rather than just writing tests with the InMemory provider because I wanted to inspect the database to be 100 percent sure that the data was getting stored the way I expected. ■

Julie Lerman *is a Microsoft Regional Director, Microsoft MVP, software team coach and consultant who lives in the hills of Vermont. You can find her presenting on data access and other topics at user groups and conferences around the world. She blogs at the 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: *Andriy Svyryd*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION 1993 - 2018

VISUAL STUDIO LIVE! (VSLive!™) is celebrating 25 years as one of the most respected, longest-standing, independent developer conferences, and we want you to be a part of it. Join us in 2018 for #VSLive25, as we highlight how far technology has come in 25 years, while looking toward the future with our unique brand of training on .NET, the Microsoft Platform and open source technologies in seven great cities across the US.

HELP US CELEBRATE #VSLIVE25. WHICH LOCATION WILL YOU ATTEND IN 2018?

APRIL 30 – MAY 4
Hyatt Regency Austin

Code Like It's 2018!

SEPTEMBER 17 – 20
Renaissance Chicago

Look Back to Code Forward.

AGENDA ANNOUNCED!

JUNE 10 – 14
Hyatt Regency Cambridge

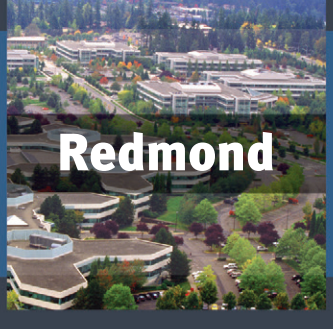
Developing Perspective.

OCTOBER 7 – 11
Hilton San Diego Resort

Code Again for the First Time!

AUGUST 13 – 17
Microsoft Headquarters

Yesterday's Knowledge; Tomorrow's Code!



DECEMBER 2 – 7
Loews Royal Pacific Resort

Code Odyssey.

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

vslive.com

#VSLIVE25

Visual Studio LIVE!

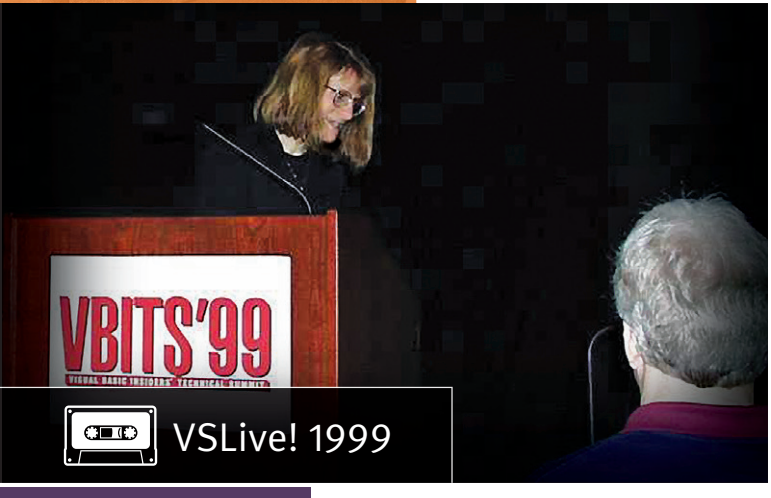
EXPERT SOLUTIONS FOR .NET DEVELOPERS

April 30 – May 4, 2018
Hyatt Regency Austin

Austin, TX

We're Gonna Code Like It's 2018!

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION 1993 - 2018



VSLive! 1999

VSLive! 2017

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

Development Topics Include:

- Visual Studio / .NET
- JavaScript / Angular
- Xamarin
- Software Practices
- Database and Analytics
- ASP.NET / Web Server
- ALM / DevOps
- Azure/Cloud
- UWP
- Hands-On Labs

REGISTER NOW

@ vslive.com/austinmsdn

Register to code with us today!

Register by April 11 and Save $200!

Use promo code AUTWO

SILVER SPONSOR

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA — YOUR GROWTH, OUR BUSINESS.

AGENDA AT-A-GLANCE

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS **Austin, TX**

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

START TIME	END TIME						
Pre-Conference Workshops: Monday, April 30, 2018 *(Separate entry fee required)*							
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries					
9:00 AM	6:00 PM	**M01** Workshop: AI, Bots, ASP.NET with DevOps - *Brian Randell*		**M02** Workshop: Web Development in 2018 - *Chris Klug*		**M03** Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka and Jason Bock*	
6:45 PM	9:00 PM	Dine-A-Round - *Speakers include Brian Randell, Jason Bock & Philip Japikse*					
START TIME	**END TIME**	**Day 1: Tuesday, May 1, 2018**					
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	**T01** TypeScript: The Future of Front End Web Development. - *Ben Hoelting*		**T02** Flying High with Xamarin! - *Sam Basu*		**T03** Entity Framework Core 2 for Mere Mortals - *Philip Japikse*	**T04** - .NET, What Does That Mean? - *Chris Klug*
9:30 AM	10:45 AM	**T05** Angular 101 - *Deborah Kurata*		**T06** Essential Tools for Xamarin Developers! - *Sam Basu*		**T07** Busy Developer's Guide to NoSQL - *Ted Neward*	**T08** - Get Started with Git - *Robert Green*
11:00 AM	12:00 PM	**KEYNOTE: From Waterfall to Agile. Microsoft's Not-So-Easy Evolution into the World of DevOps** - *Abel Wang, Sr. Developer Technology Specialist, Microsoft*					
12:00 PM	1:00 PM	Lunch					
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors					
1:30 PM	2:45 PM	**T09** The Many Adventures of Routing - *Deborah Kurata*		**T10** Busy Developer's Guide to the Clouds - *Ted Neward*		**T11** Power BI for Developers - *Thomas LeBlanc*	**T12** Building Apps with Microsoft Graph and Visual Studio - *Robert Green*
3:00 PM	4:15 PM	**T13** ASP.NET Core 2 For Mere Mortals - *Philip Japikse*		**T14** Cloud-Oriented Programming - *Vishwas Lele*		**T15** SQL Server 2017 Execution Plan Basics - *Thomas LeBlanc*	**T16** Advanced DevOps—Deep Dive into Feature Flags - *Abel Wang*
4:15 PM	5:30 PM	Welcome Reception					
START TIME	**END TIME**	**Day 2: Wednesday, May 2, 2018**					
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	**W01** Assembling the Web—A Tour of WebAssembly - *Jason Bock*		**W02** - .NET Core, Containers, and K8S in Azure - *Brian Randell*		**W03** New SQL Server Security Features for Developers - *Leonard Lobel*	**W04** Works On My Machine... Docker for Microsoft Developers - *Chris Klug*
9:30 AM	10:45 AM	**W05** Tools for Modern Web Development - *Ben Hoelting*		**W06** Microservices with ACS (Managed Kubernetes) - *Vishwas Lele*		**W07** Introduction to Azure Cosmos DB - *Leonard Lobel*	**W08** Professional Scrum Development Using Visual Studio- *Richard Hundhausen*
11:00 AM	12:00 PM	**GENERAL SESSION: .NET in 2018** - *Scott Hunter, Partner Director Program Management, .NET, Microsoft*					
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	1:50 PM	**W09** Fast Focus: Living Happily Together: Visual Studio and Angular - *Brian Noyes*		**W10** Fast Focus: JSON Integration in SQL Server - *Leonard Lobel*		**W11** Fast Focus: Getting Git - *Jason Bock*	
2:00 PM	2:20 PM	**W12** Fast Focus: Getting Started with ASP.NET Core 2.0 Razor Pages - *Walt Ritscher*		**W13** Fast Focus: Xamarin.Forms Takes You Places! - *Sam Basu*		**W14** Fast Focus: Reuse Your .NET Code Across Platforms - *Rockford Lhotka*	
2:30 PM	3:45 PM	**W15** Introduction to HTML5 - *Paul Sheriff*		**W16** Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*		**W17** Writing Testable Code and Resolving Dependencies—DI Kills Two Birds with One Stone - *Miguel Castro*	**W18** DevOps for the SQL Server Database - *Brian Randell*
4:00 PM	5:15 PM	**W19** Introduction to CSS 3 - *Paul Sheriff*		**W20** Lock the Doors, Secure the Valuables, and Set the Alarm - *Eric D. Boyd*		**W21** Dependencies Demystified - *Jason Bock*	**W22** Use Visual Studio to Scale Agile in Your Enterprise - *Richard Hundhausen*
6:45 PM	9:00 PM	VSLive! Rollin' On the River Bat Cruise					
START TIME	**END TIME**	**Day 3: Thursday, May 3, 2018**					
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries					
8:00 AM	9:15 AM	**TH01** Securing Angular Apps - *Brian Noyes*		**TH02** Use UWP to Modernize Your Existing WinForms and WPF Applications - *Walt Ritscher*		**TH03** Coaching Skills for Scrum Masters & The Self-Organizing Team - *Benjamin Day*	**TH04** Improving Code Quality with Roslyn Code Analyzers - *Jim Wooley*
9:30 AM	10:45 AM	**TH05** Securing Web Apps and APIs with IdentityServer - *Brian Noyes*		**TH06** A Dozen Ways to Mess Up Your Transition From Windows Forms to XAML - *Billy Hollis*		**TH07** Demystifying Microservice Architecture - *Miguel Castro*	**TH08** Getting to the Core of the .NET Standard - *Adam Tuliper*
11:00 AM	12:15 PM	**TH09** ASP Core Middleware & More - *Miguel Castro*		**TH10** Building Cross Platform Business Apps with CSLA .NET - *Rockford Lhotka*		**TH11** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**TH12** C# 7.x Like a Boss! - *Adam Tuliper*
12:15 PM	1:30 PM	Lunch					
1:30 PM	2:45 PM	**TH13** Getting Pushy with SignalR and Reactive Extensions - *Jim Wooley*		**TH14** UX Design for Developers: Basics of Principles and Process - *Billy Hollis*		**TH15** Modernize Your Legacy Applications with Continuous Integration and Continuous Delivery - *David Walker*	**TH16** Scale Agile without Friction: DevOps to the Rescue - *Clementino de Mendonca*
3:00 PM	4:15 PM	**TH17** Migrating to ASP.NET Core—A True Story - *Adam Tuliper*		**TH18** Add a Conversational User Interface to Your App with the Microsoft Bot Framework - *Walt Ritscher*		**TH19** Hard-Core Integration Intoxication with .NET Core - *David Walker*	**TH20** VSTS as a Knowledge Management Hub: Secure the Future of Your IP - *Clementino de Mendonca*
START TIME	**END TIME**	**Post-Conference Full Day Hands-On Labs: Friday, May 4, 2018** *(Separate entry fee required)*					
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries					
8:00 AM	5:00 PM	**HOL01** Full Day Hands-On Lab: Develop an ASP.NET Core2 and EF Core 2 App in a Day - *Philip Japikse*				**HOL02** Full Day Hands-On Lab: Developer Dive into SQL Server - *Leonard Lobel*	

Speakers and sessions subject to change

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/austinmsdn

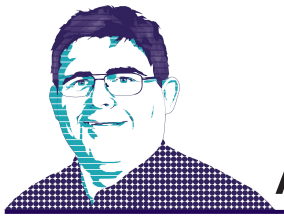
AGENDA AT-A-GLANCE

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

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

START TIME	END TIME				
		Pre-Conference Full Day Hands-On Labs: Sunday, June 10, 2018 *(Separate entry fee required)*			
7:00 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries			
8:00 AM	5:00 PM	**HOL01** Full Day Hands-On Lab: Develop an ASP.NET Core 2 and EF Core 2 App in a Day - *Philip Japikse*		**HOL02** Full Day Hands-On Lab: From 0-60 in a Day with Xa marin and Xamarin.Forms - *Roy Cornelissen & Marcel de Vries*	
START TIME	END TIME				
		Pre-Conference Workshops: Monday, June 11, 2018 *(Separate entry fee required)*			
8:00 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	**M01** Workshop: DevOps, You Keep Saying That Word - *Brian Randell*	**M02** Workshop: SQL Server for Developers - *Andrew Brust and Leonard Lobel*	**M03** Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka and Jason Bock*	
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	END TIME				
		Day 1: Tuesday, June 12, 2018			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**T01** Angular 101 - *Deborah Kurata*	**T02** A Developer's Introduction to XR: Virtual, Augmented & Mixed Reality - *Nick Landry*	**T03** SQL Server 2017 - Intelligence Built-in - *Scott Klein*	**T04** Getting to the Core of the .NET Standard - *Adam Tuliper*
9:30 AM	10:45 AM	**T05** ASP.NET Core 2 For Mere Mortals - *Philip Japikse*	**T06** Lessons Learned from Real-World HoloLens & Mixed Reality Projects - *Nick Landry*	**T07** Bots and AI with Azure - *Scott Klein*	**T08** To Be Announced
11:00 AM	12:00 PM	**KEYNOTE: To Be Announced**			
12:00 PM	1:00 PM	Lunch			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors			
1:30 PM	2:45 PM	**T09** Migrating to ASP.NET Core - A True Story - *Adam Tuliper*	**T10** Re-imagining an App's User Experience: A Real-world Case Study - *Billy Hollis*	**T11** Introduction to Azure Cosmos DB - *Leonard Lobel*	**T12** What's New in C#7 - *Jason Bock*
3:00 PM	4:15 PM	**T13** Angular Component Communication - *Deborah Kurata*	**T14** There is No App - *Roy Cornelissen*	**T15** SQL Server Security Features for Developers - *Leonard Lobel*	**T16** Concurrent Programming in .NET - *Jason Bock*
4:15 PM	5:30 PM	Welcome Reception			
START TIME	END TIME				
		Day 2: Wednesday, June 13, 2018			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** An Introduction to TypeScript - *Jason Bock*	**W02** Strategies and Decisions for Developing Desktop Business Applications - *Billy Hollis*	**W03** Applying ML to Software Development - *Vishwas Lele*	**W04** Architecting Systems for Continuous Delivery - *Marcel de Vries*
9:30 AM	10:45 AM	**W05** To Be Announced	**W06** Cross-Platform App Dev with Xamarin and CSLA .NET - *Rockford Lhotka*	**W07** Predicting the Future Using Azure Machine Learning - *Eric D. Boyd*	**W08** Database DevOps - *Brian Randell*
11:00 AM	12:00 PM	**GENERAL SESSION: .NET Everywhere and for Everyone** - *James Montemagno, Principal Program Manager - Mobile Developer Tools, Microsoft*			
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	**W09** User Authentication for ASP.NET Core MVC Applications - *Brock Allen*	**W10** Xamarin: The Future of App Development - *James Montemagno*	**W11** Analytics and AI with Azure Databricks - *Andrew Brust*	**W12** Using Feature Toggles to Help Us Seperate Releases from Deployments - *Marcel de Vries*
3:00 PM	4:15 PM	**W13** Securing Web APIs in ASP.NET Core - *Brock Allen*	**W14** Programming with the Model-View-ViewModel Pattern - *Miguel Castro*	**W15** Power BI: What Have You Done for Me Lately? - *Andrew Brust*	**W16** Azure DevOps with VSTS, Docker, and K8 - *Brian Randell*
4:30 PM	5:45 PM	**W17** Multi-targeting the World - *Oren Novotny*	**W18** Cognitive Services in Xamarin Applications - *Veronika Kolesnikova & Willy Ci*	**W19** Azure in the Enterprise - *Mike Benkovich*	**W20** Visualizing the Backlog with User Story Mapping - *Philip Japikse*
6:15 PM	8:30 PM	VSLive!'s Boston By Land and Sea Tour			
START TIME	END TIME				
		Day 3: Thursday, June 14, 2018			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** Tools for Modern Web Development - *Ben Hoelting*	**TH02** Writing Testable Code and Resolving Dependencies - DI Kills Two Birds with One Stone - *Miguel Castro*	**TH03** Containers Demystified - *Robert Green*	**TH04** Busy .NET Developer's Guide to Python - *Ted Neward*
9:30 AM	10:45 AM	**TH05** JavaScript Patterns for the C# Developer - *Ben Hoelting*	**TH06** Netstandard: Reuse Your Code on Windows, Linux, Mac, and Mobile - *Rockford Lhotka*	**TH07** Microservices with ACS (Managed Kubernetes) - *Vishwas Lele*	**TH08** Signing Your Code the Easy Way - *Oren Novotny*
11:00 AM	12:15 PM	**TH09** ASP Core Middleware & More - *Miguel Castro*	**TH10** The Role of an Architect - *Ted Neward*	**TH11** Go Serverless with Azure Functions - *Eric D. Boyd*	**TH12** Get Started with Git - *Robert Green*
12:15 PM	1:15 PM	Lunch			
1:15 PM	2:30 PM	**TH13** Enhancing Web Pages with VueJS: When You Don't Need a Full SPA - *Shawn Wildermuth*	**TH14** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**TH15** Computer, Make It So! - *Veronika Kolesnikova & Willy Ci*	**TH16** C# 7, Roslyn and You - *Jim Wooley*
2:45 PM	4:00 PM	**TH17** Versioning APIs with ASP.NET Core - *Shawn Wildermuth*	**TH18** Coaching Skills for Scrum Masters & The Self-Organizing Team - *Benjamin Day*	**TH19** Compute Options in Azure - *Mike Benkovich*	**TH20** Improving Code Quality with Roslyn Code Analyzers - *Jim Wooley*

Speakers and sessions subject to change



FRANK LA VIGNE

Introducing Apache Spark ML

Apache Spark ML is a machine learning library module that runs on top of Apache Spark. Spark itself is a cluster computing environment that provides data engineers and data scientists an interface for programming entire clusters of commodity computers with data parallelism and fault tolerance. Spark supports a number of languages such as Java, Scala, Python and R. It also natively provides a Jupyter Notebook service. Please refer to my February Artificially Intelligent column (msdn.com/magazine/mt829269) for the fundamentals of Jupyter Notebooks if you're not familiar with them. In this article, I'll explore using Spark ML in a Juptyer Notebook on an HDInsight cluster on Azure.

Getting Started with Apache Spark on Azure

To work with Spark ML, the first step is to create a Spark cluster. Log into the Azure Portal and choose "Create a resource," then choose HDInsight, as shown in **Figure 1**. The blades that appear walk through the process of creating an HDInsight cluster.

The first blade, labeled Basics, covers essential properties of the cluster, such as Cluster Name, administrator and ssh credentials, as well as resource group and location. The cluster name must be unique across the azurehdinsight.net domain. For reference, please refer to **Figure 2**. Of particular importance is the Cluster Type option, which brings up yet another blade. In this blade, shown in **Figure 3**, set the cluster type to Apache Spark and the version to Spark 2.10 (HDI 3.6). Click Select to save the setting and close the blade.

The next step configures storage options for the cluster. Leave Primary Storage Type and Selection Method at their defaults. For storage container, click create new and name it "msdnsparkstorage" and set the default container to "sparkstorage" in the Default Container textbox. Click Next to get to the Summary step of the setup process. This screen offers a review and opportunity to modify the cluster setup along with the estimated hourly cost to run the cluster. Take special care to always delete clusters when not in use. Unlike virtual machines in Azure, HDInsight clusters do not have an option to pause and stop billing. Click Create and take note of the notification that it can take up to 20 minutes to instantiate the cluster.

> When running HDInsight clusters, Azure Storage Blobs seamlessly map to HDFS.

HDFS and Azure Blob Storage

Spark, like Hadoop, uses the Hadoop Distributed File System (HDFS) as its cluster-wide file store. HDFS is designed to reliably store large datasets and make those datasets rapidly accessible to applications running on the cluster. As the name implies, HDFS originated within Hadoop and is also supported by Spark.

When running HDInsight clusters, Azure Storage Blobs seamlessly map to HDFS. This fact makes it simple to upload and download data to and from the HDFS file store attached to the Spark cluster. In fact, the first step to getting the project started will be using Azure Storage Explorer to upload the data file with which to work.

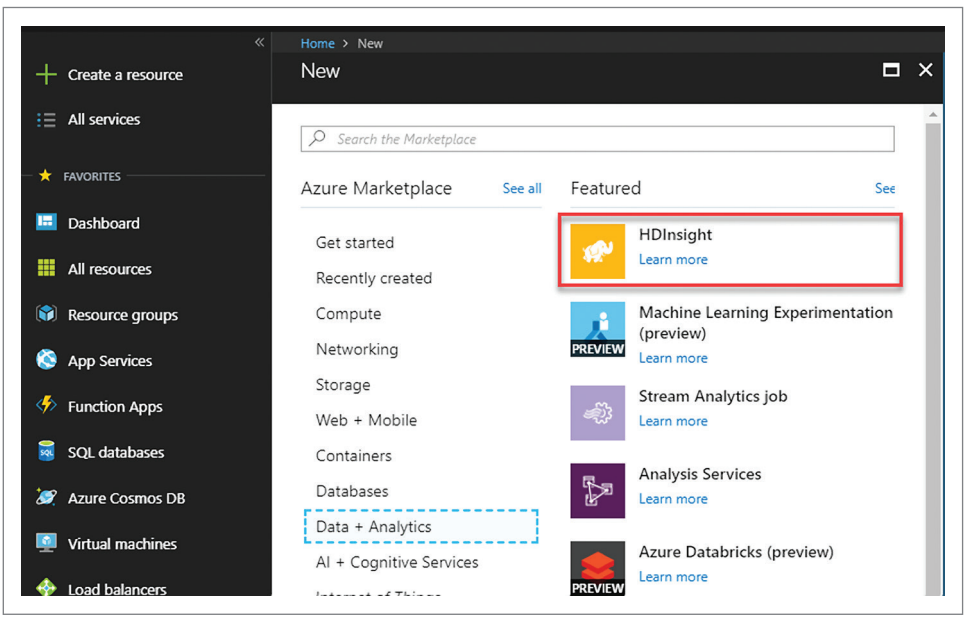


Figure 1 **Creating a New HDInsight Resource**

Azure Storage Explorer is a free, cross-platform utility to manage data stores in Azure. If you don't already have it installed, please do so while waiting for the Spark cluster to initialize (azure.microsoft.com/features/storage-explorer).

Once the cluster is set up and Azure Storage Explorer is configured to access the storage account created for the cluster in **Figure 2**, open Azure Storage Explorer and browse to the "msdndemo" blob using the tree control on the left (**Figure 4**). Click on "msdndemo" to reveal the contents at the root of the blob, then click New Folder and in the Create New Virtual Directory dialog, enter the name for the new folder: flights. Click OK to create the folder. Next, click on the Upload button, choose Upload Files, click the ellipsis button, and browse to the CSV data file for this article, "06-2015.csv." Click Upload to upload the file to the Blob store.

Now that the data file is uploaded, it's time to start working with the file in a PySpark Notebook. The Spark Python API, commonly referred to as PySpark, exposes the Spark programming model to Python. For developers accustomed to Python, PySpark will feel very familiar. The Spark Web site provides a great introductory explanation to the environment and how it differs from standard Python (bit.ly/2oVBuCy).

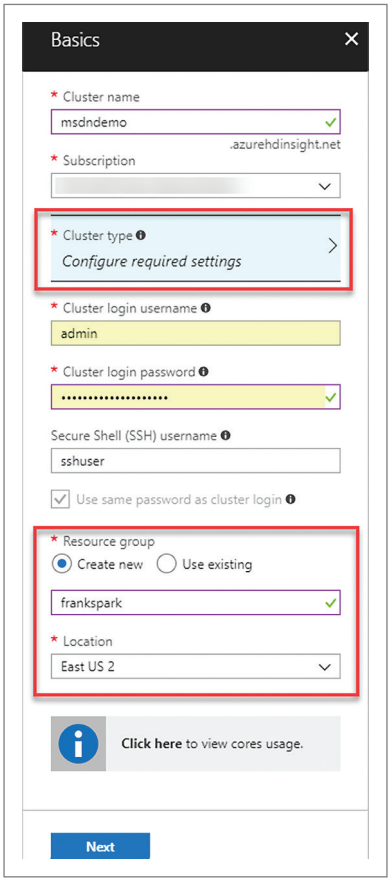


Figure 2 **The Basics Step of the Quick Create Process for Setting Up an HDInsight Cluster**

Jupyter Notebooks in Spark

The HDInsight implementation of Apache Spark includes an instance of Jupyter Notebooks already running on the cluster. The easiest way to access the environment is to browse to the Spark cluster blade on the Azure Portal. On the overview tab, click either of the items labeled Cluster Dashboard (**Figure 5**). In the blade that appears, click on the Jupyter Notebook tile. If challenged for credentials, use the cluster login credentials created earlier.

Once the homepage for the cluster's Jupyter Notebooks service loads, click New and then choose PySpark3 to create a new PySpark3 notebook, as depicted in **Figure 6**.

Doing this will create a new blank notebook with an empty cell. In this first cell, enter the following code to load the CSV file uploaded to the blob earlier. Before pressing Control+Enter on the keyboard to execute the code, examine the code first:

```
flights_df = spark.read.csv('wasb:///flights/06-2015.csv',
  inferSchema=True, header=True)
```

Note the "wasb" protocol handler in the URL. WASB stands for Windows Azure Storage Blobs and provides an interface between Hadoop and Azure Blob storage. For more information about how this was done and why this is significant, refer to the blog posts, "Why WASB Makes Hadoop on Azure So Very Cool" (bit.ly/2oUXptz), and "Understanding WASB and Hadoop Storage in Azure" (bit.ly/2ti43zu), both written by Microsoft developer Cindy Gross. For now, however, the key takeaway is that Azure Blobs can act as persistent file stores for data even when the Spark cluster isn't running. Furthermore, the data stored here is accessible by applications that support either Azure Blob storage or HDFS.

With the focus still on this textbox, press Control+Enter. Immediately, the space beneath the cell will read "Starting Spark application." After a moment, a table appears with some data about the job that just ran and a notification that the "SparkSession is available as 'spark.'" The parameters passed along to the spark.read.csv method automatically inferred a schema and indicated that the file has a header row. The contents of the CSV file were loaded into a DataFrame object. To view the schema, enter the following code into the newly created blank cell and then execute the code by pressing Control+Enter:

```
flights_df.printSchema()
```

The schema appears and displays the name and datatype of each field. The names match the names of the fields from the header row of the CSV file.

DataFrames in Detail

In Spark, DataFrames are a distributed collection of rows with named columns. Practically speaking, DataFrames provide an interface similar to tables in relational databases or an Excel worksheet sheet with named column headers. Under the covers, DataFrames provide an API to a lower level fundamental data structure in Spark: a resilient distributed dataset (RDD). RDDs are a fault-tolerant, immutable, distributed collection of objects that can be worked on in parallel across the worker nodes in a Spark cluster.

RDDs themselves are divided into smaller pieces called Partitions. Spark automatically determines the number of partitions into which to split an RDD. Partitions are distributed across the nodes on the cluster. When an action is performed on an RDD, each of its partitions launches a task and the action is executed in parallel. Happily, for those new to Spark, most of the architecture is abstracted away: first by the RDD data structure and Spark, then by the higher-level abstraction of the DataFrame API.

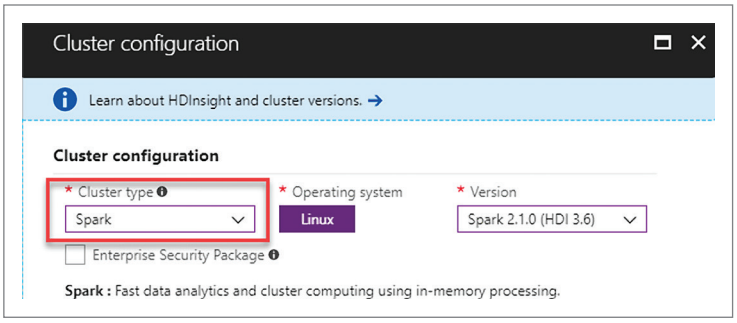


Figure 3 **Choosing Spark as the Cluster Type for the HDInsight Cluster**

Exploring Data with DataFrames

DataFrames expose multiple ways to explore data. For instance, to view the shape of the data, enter the following code into a blank cell and then execute the code:

```
recordCount = flights_df.count()
column_count = len(flights_df.columns)
print ("DataFrame has " + str(recordCount) + "rows and " + str(
  column_count) + " columns.")
```

The resulting response will read:

```
DataFrame has 503897 rows and 26 columns.
```

> The heavy lifting to parallelize the execution of the data queries is done by the Spark core engine.

To view only the flights leaving BWI airport, enter the following into an empty cell and execute:

```
flights_leaving_BWI = flights_df.where(flights_df.ORIGIN == 'BWI')
flights_leaving_BWI.show()
```

The code created a new DataFrame in the flights_leaving_BWI with results filtered from the flights_df DataFrame. The show method displays the top 20 rows. The results may look a bit disorganized due to the nature of the formatting constraints of the Web page and the DataFrame containing 26 columns. Right now, I just want to see the airline carrier, flight number, day of month and flight destination. To do this, modify the code in the cell to the following and execute again:

```
flights_leaving_BWI = flights_df.where(flights_df.ORIGIN == 'BWI').
select("UNIQUE_CARRIER",
  "FL_NUM", "TAIL_NUM",  "DAY_OF_MONTH", "DEST")
flights_leaving_BWI.show()
```

The formatting will look better as there are only five columns returned and will fit better onto the Web page. What if you wanted to sort the results by flight destination and show the first 100 rows as opposed to the first 20? Change the code in the cell to the following to do just that:

```
flights_leaving_BWI = flights_df.where(flights_df.ORIGIN == 'BWI').
select("UNIQUE_CARRIER",
  "FL_NUM", "TAIL_NUM",  "DAY_OF_MONTH", "DEST").sort("DEST")
flights_leaving_BWI.show(100)
```

What if you wanted to create a DataFrame of all the flights departing the DC area's three major airports? To do that, enter the following code into a new cell and execute:

```
flights_leaving_DCmetro = flights_df.where( (flights_df.ORIGIN == 'BWI') |
  (flights_df.ORIGIN == 'DCA') | (flights_df.ORIGIN == 'IAD') ).select(
  "UNIQUE_CARRIER", "FL_NUM", "TAIL_NUM",  "DAY_OF_MONTH", "ORIGIN", "DEST")
flights_leaving_DCmetro.show()
```

Note that I added the origin field into this DataFrame to show from where a flight originated.

Create a new cell, enter the following code and execute it:

```
print ("Flights leaving BWI: " + str(flights_leaving_BWI.count()))
print ("Flights leaving DC Metro: " + str(flights_leaving_DCmetro.count()))
```

The results returned should look something like this:

```
Flights leaving BWI: 8422
Flights leaving DC Metro: 18502
```

As you can see, working with DataFrames is intuitive and simple. The heavy lifting to parallelize the execution of the data queries is done by the Spark core engine. Additionally, anyone familiar with Hive or Pig will notice that execution time on these queries is significantly faster in Spark.

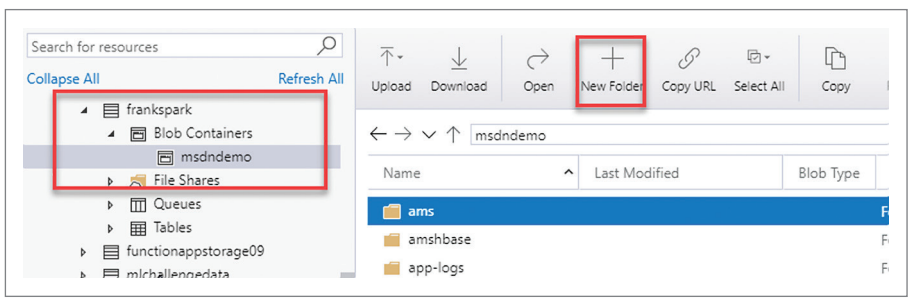


Figure 4 **Creating a New Folder in the Blob Container Created for the Spark Cluster**

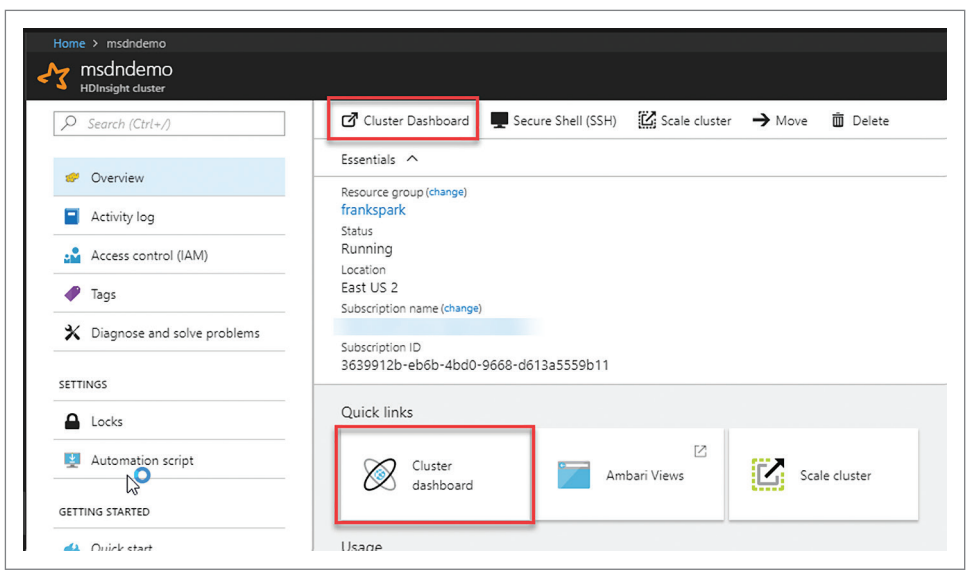


Figure 5 **Launching the Jupyter Notebooks Environment for the Spark Cluster**

Creating a Predictive Model with SparkML

To create a predictive model with Spark ML, I need to import some libraries into my project. To do this, create a new cell, enter the following code, and execute it:

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml import Pipeline
from pyspark.ml.feature import
VectorAssembler

from pyspark.ml.classification
import LogisticRegression
```

Next, I need to create a new DataFrame with just the fields I need to create a predictive model of the likelihood of a flight being delayed. The following code pares down the original 26 columns in

DevExpress DXperience 17.2 | from $1,439.99

DevExpress

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

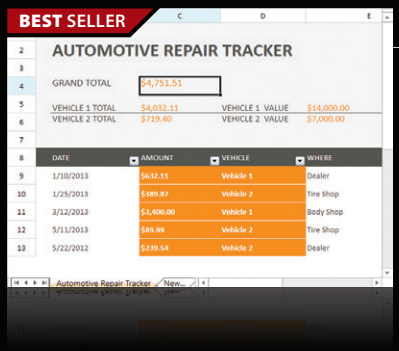
- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context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

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

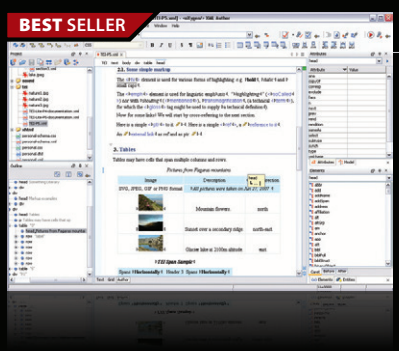


Spread.NET | from $1,476.52

GrapeCity

Deliver multi-functional spreadsheets in less time with Visual Studio.

- Designed from the ground up for maximum performance and speed
- Gives you the features you expect for seamless Excel compatibility with NO Excel dependency
- Powerful calculation engine handles huge numbers of rows and columns with ease
- Use the extensive API to control every aspect of workbooks, worksheets, ranges, and cells
- Benefit from the integrated calculation engine with more than 462 Excel functions available



Oxygen XML Editor Professional | from $660.48

syncro soft

The complete XML editing solution, both for XML developers and content authors

- Produce output in PDF, ePUB, HTML, and many other formats from a single source
- Ready-to-use support for DITA, DocBook, XHTML, and TEI frameworks
- Extend the built-in XML publishing frameworks, or even create your own frameworks
- Interact with the majority of XML databases, content management systems, and WebDAV
- Make sure your XML documents are "well-formed" and valid, using as-you-type validation

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

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

Europe (UK)
ComponentSource
The White Building
33 King's Road
Reading, Berkshire
RG1 3AR
United Kingdom

Europe (Ireland)
ComponentSource
Unit 1E & 1F
Core B, Block 71
The Plaza, Park West
Dublin 12
Ireland

Asia Pacific
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

the DataFrame to just six. While the original dataset has a column indicating if a flight was delayed 15 minutes or more, I would like to add some more granularity. I'll create a new column set to 1 if a flight is delayed 10 minutes or more and name that column "IsDelayed." Additionally, I call the dropna function, which drops any rows with null values. ML algorithms can be finicky about the data inputs they receive. Very often an unexpected null value will throw an exception, or worse, corrupt the results. Enter the following code into a new cell and execute it. The results will show the first 20 rows of model_data:

```
model_data = flights_df.select("DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN_AIRPORT_ID",
  "DEST_AIRPORT_ID", "DEP_DELAY", ((col("ARR_DELAY") >
  10).cast("Int").alias("IsDelayed"))).dropna()

  model_data.show()
```

Supervised Learning

In supervised ML, the ground truth is known. In this case, it's the on-time arrival records of flights in the United States for June 2015. Based on that data, the algorithm will generate a predictive model on whether a flight's arrival will be delayed by 10 minutes or more given the five fields: Day of Month, Day of Week, Origin Airport ID, Destination Airport ID and Departure Delay. In ML terms, those fields are collectively referred to as "features." The predicted value, in this case an indicator of an arrival delay of 10 minutes or less than 10 minutes, is referred to as the "label."

Supervised ML models are often tested with the same dataset on which they're trained. To do this, they're randomly split into two datasets: one for training and one for testing. Usually, they're split along a 60/40 or 70/30 line with the larger share going into the training data.

The following code separates the training data and the test data into two sets along a 70/30 line and displays the count (once again, enter the code into a new cell and execute it):

```
split_model_data = model_data.randomSplit([0.7,0.3])
training_data = split_model_data[0]
testing_data = split_model_data[1]

all_data_count = training_data.count() + testing_data.count()

print ("Rows in Training Data: " + str(training_data.count()))
print ("Rows in Testing Data: " + str(testing_data.count()))
print ("Total Rows: " + str(all_data_count))
```

There will be a discrepancy in the total rows displayed here and earlier in the notebook. That's due to rows with null values being dropped.

The test data now must be further modified to meet the requirements of the ML algorithm. The five fields representing the features will be combined into an array, or a vector, through a process called vectorization. The IsDelayed column will be renamed to label. The training DataFrame will have just two columns: features and label. Enter the following code into an empty cell and execute it and the first 20 rows of the training DataFrame will be displayed:

```
vector_assembler = VectorAssembler(
  inputCols = ["DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN_AIRPORT_ID", "DEST_AIRPORT_ID",
  "DEP_DELAY"], outputCol="features")
training = vector_assembler.transform(training_data).select(col("features"),
  col("IsDelayed").cast("Int").alias("label"))
  training.show(truncate=False)
```

> Supervised ML models are often tested with the same dataset on which they're trained.

With the training data split into two columns, features and labels, it's ready to be fed to the ML algorithm. In this case, I've chosen logistic regression. Logistic regression is a statistical method for analyzing data where one or more input variables influence an outcome. For this model, the input variables are the contents of the feature column, the fields DAY_OF_MONTH, DAY_OF_WEEK, ORIGIN_AIRPORT_ID, DEST_AIRPORT_ID and DEP_DELAY. The outcome is the label column, or if the flight was delayed more than 10 minutes. This algorithm will not distinguish between a delay of 10 minutes and one second and a 15-hour delay. The model is created by fitting the training data to it. Again, enter the following code into a blank cell and execute it:

```
logR = LogisticRegression(labelCol="label", featuresCol="features",
  maxIter=10, regParam=0.1)
model = logR.fit(training)
```

With the model trained, the only thing left to do is test it. The testing data must also be modified to fit the expectations of the algorithm by running it through the vector assembler as the training data was. Enter the following code into a blank cell and execute it:

```
testing = vector_assembler.transform(testing_data).select(col("features"),
  (col("IsDelayed")).cast("Int").alias("trueLabel"))
  testing.show()
```

Now that the training data is prepared, the next step is to run it through the model by calling the transform method. The output is a DataFrame with four columns: the features, the predicted value, the actual value and the probability, a measure of how confident the algorithm was in its prediction. Once more, enter the following code into an empty cell and execute it:

```
prediction = model.transform(testing)
predicted = prediction.select("features",
"prediction", "trueLabel", "probability")
predicted.show()
```

The output only shows the first 20 rows. That's not an efficient way of measuring

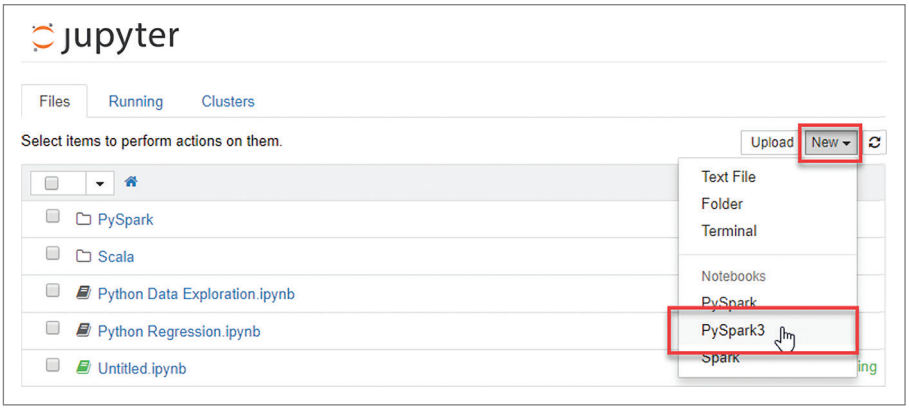


Figure 6 **Creating a New PySpark 3 Jupyter Notebook**

the efficacy of the model. The best way to do that is to count the number of times the algorithm predicted correctly and how many times it was wrong. However, a simple "right and wrong" metric doesn't always tell the full story. The better metric is something called the "confusion matrix," which displays the number of true negatives and true positives along with the number of false positives and false negatives. Enter the following code into a blank cell and execute it to display the confusion matrix for this model:

```
true_negatives = predicted.where( (predicted.prediction == '0.0') |
  (predicted.trueLabel == '0')).count()
true_positives = predicted.where( (predicted.prediction == '1.0') |
  (predicted.trueLabel == '1')).count()
false_negatives = predicted.where( (predicted.prediction == '0.0') |
  (predicted.trueLabel == '1')).count()
false_positives = predicted.where( (predicted.prediction == '1.0') |
  (predicted.trueLabel == '0')).count()

print ( "True Positive: " + str(true_positives) )
print ( "True Negative: " + str(true_negatives) )
print ( "False Positive: " + str(false_positives) )
print ( "False Negative: " + str(false_negatives) )
```

When experiments fail, the best course of action is to analyze the results, tweak the model parameters and try again.

The results are not encouraging. The model was wrong considerably more than it was right. All is not lost, however. When experiments fail, the best course of action is to analyze the results, tweak the model parameters and try again. This is why the field is called "data science."

Wrapping Up

Spark is a fast and powerful cluster computing environment for parallel processing of data workloads with a modular architecture. Two modules explored in this article were PySpark and Spark ML. PySpark provides a run Python runtime for Spark and high-level abstraction of Resilient Distributed Datasets (RDDs) in the form of a DataFrames API. The Spark ML library provides a machine learning API for data built on top of DataFrames.

Machine learning is a discipline with the larger field of data science. When an experiment doesn't yield the desired results, finding the solution requires an iterative approach. Perhaps 10 minutes is too granular an interval. Maybe more than five input fields would help uncover a pattern. Quite possibly one month's worth of flight data is not enough for the algorithm to establish a clear pattern. The only way to know is to keep experimenting, analyzing the results, and adjusting the input data and parameters. ■

Frank La Vigne *leads the Data & Analytics practice at Wintellect and co-hosts the DataDriven podcast. He blogs regularly at FranksWorld.com and you can watch him on his YouTube channel, "Frank's World TV" (FranksWorld.TV).*

Thanks to the following technical expert for reviewing this article: *Andy Leonard*

Deploying Containerized ASP.NET Core Applications to Azure

Srikatan Sankaran

Azure Key Vault is a cloud-based service that organizations can leverage to securely store sensitive business information. In the last issue of *MSDN Magazine* (msdn.com/magazine/mt845653), I explored a scenario in which insurance policies for vehicles generated by an agency could be stored securely in Azure Key Vault as secrets, and showed how additional features like asymmetric encryption with keys and built-in versioning could be used to maintain an audit trail of access to insurance policies. The applications that used Azure Key Vault were deployed to Azure App Service as Web apps. In this issue, I'll be deploying the same applications to Azure Container Services for Kubernetes (AKS). Azure Container Services provides the ability to create and deploy Docker container-enabled applications to Azure.

Some of the products discussed in this article are in preview, so all information is subject to change.

This article discusses:

- Using Azure Container Registry to store and pull Docker containers
- Using Visual Studio 2017 Tools to create Docker images
- Using kubectl tools for Kubernetes to deploy containerized applications to Azure Container Service – AKS
- Using Azure CLI to push and pull container images to and from Azure Container Registry

Technologies discussed:

Azure Container Services for Kubernetes, Docker, Azure Application Gateway, ASP.NET Core 2.0, Azure Container Registry, Azure Active Directory, Azure Team Services, Visual Studio 2017 Tools for Docker, Azure Container Instance, Azure Web App for Containers

Code download available at:

bit.ly/2DRvwdh

Use-Case Scenario

There are two ASP.NET Core 2.0 Web applications in the solution accompanying this article, an admin portal used by the insurance companies, agencies and brokers, and a customer portal used by either the consumers who purchase the vehicle insurance policies or by regulatory authorities to ascertain the validity of the insurance policy. Refer to the previous article for more details on the business scenario addressed by the solution, and to learn how Azure Key Vault helped me meet those requirements.

Architecture of the Solution

Figure 1 represents the architecture of the solution, with minor changes from the previous article. Instead of Azure App Service, a single AKS cluster is used to host both Web applications on Linux, using Docker containers. Azure Application Gateway is used to expose an SSL-enabled endpoint to clients running in the browser and to the native Xamarin app, and routes requests to the Web applications deployed in AKS. Azure Active Directory (Azure AD)

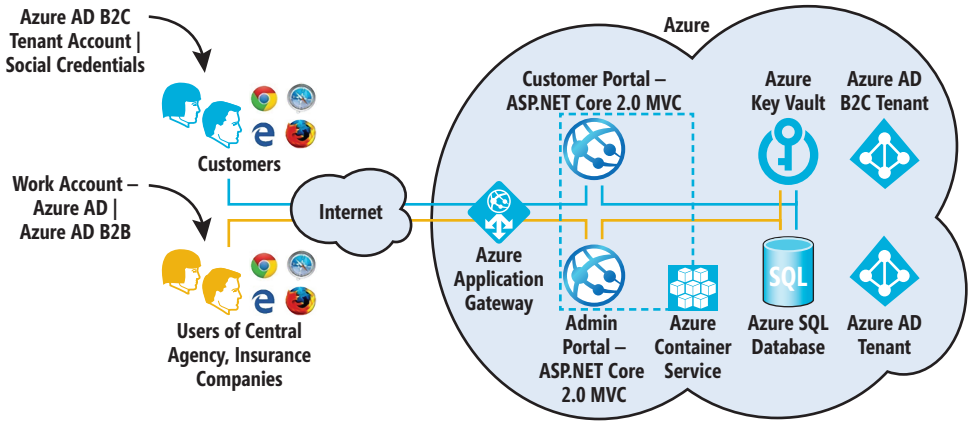


Figure 1 **Solution Architecture**

is used to authenticate the users of the admin portal while Azure Active Directory B2C (Azure AD B2C) is used for the consumers accessing the customer portal. Azure SQL Database stores the metadata information pertaining to the insurance policies created in the solution, and Azure Key Vault holds the sensitive information that will be used to ascertain the validity and authenticity of each insurance policy. There are some other options, apart from AKS, for deploying containerized applications, like Azure Container Instances (ACI) and Azure Web Apps for Containers, which are also covered in this article, though not depicted in **Figure 1**.

> ACI provides the quickest way to run a container-based application in Azure.

Modifying the Solution to Suit the New Architecture

Of course, changes to the architecture mean the solution must change. Let's take a look at how I modified the solution.

Enabling the Web Applications for Docker Containers Visual Studio Tools for Docker provides a turnkey implementation to enable an ASP.NET 2.0 Core Application for both Windows and Linux Containers. Installing this tool gives you a context-driven menu option in your Visual Studio 2017 project that generates all the artifacts required to create a Docker container for the application. The admin and consumer (external) portal applications in the solution were packaged for Docker Linux containers. Before you can use the tool in Visual Studio 2017, you must install Docker for Windows and perform the configuration steps described at bit.ly/2F6YBRb. I created the Docker container images using the "Release" configuration in Visual Studio 2017 for each of the Web application projects.

Running the Docker commands in the console windows displays two container images, one for each portal application, as shown in **Figure 2**.

Uploading Container Images to a Private Registry The container images created earlier were first uploaded to an Azure Container Registry (ACR), from where they would be consumed in Azure Container Services. The ACR instance created for this deployment is "contosoinsacr." After signing into the registry using the Azure CLI, I executed Docker commands to upload the container images to this registry:

```
az acr login --name contosoinsacr --username <username> --password <password>
docker tag contosoinsauthorityadminportal contosoinsacr.azurecr.io/
contosoinsauthorityadminportal:web1v1
docker push contosoinsacr.azurecr.io/contosoinsauthorityadminportal:web1v1
docker tag contosoinsextportal:latest contosoinsacr.azurecr.io/
contosoinsextportal:web2v1
docker push contosoinsacr.azurecr.io/contosoinsextportal:web2v1
az acr repository list --name contosoinsacr --output table
```

The final Docker command lists the container images from the ACR after they were uploaded. Refer to the documentation at bit.ly/2BZ1rdv, which details how to create an ACR and push a Docker image to it.

Deploying Containerized Web Applications

With the admin and external (customer) portal applications packaged as containers, there are multiple options available in Azure to deploy them. Let's take a look.

Deploying to Azure Container Instances ACI provides the quickest way to run a container-based application in Azure, without your having to provision any virtual machines to host them or deal with container orchestration to handle more complex scenarios. You simply use the Azure Portal experience to create a new ACI and provide the access credentials and reference to the container images uploaded to the ACR in the steps covered earlier. On completion, the container instance gets provisioned and assigned a public IP address for accessing the Web application. These steps would be performed for both the admin and customer portal applications. Refer to the documentation available bit.ly/2CmuNhK to use the Azure Portal to create an ACI and deploy an application to it for more information.

Before these applications can be used, you need to configure the "redirect url" in the application

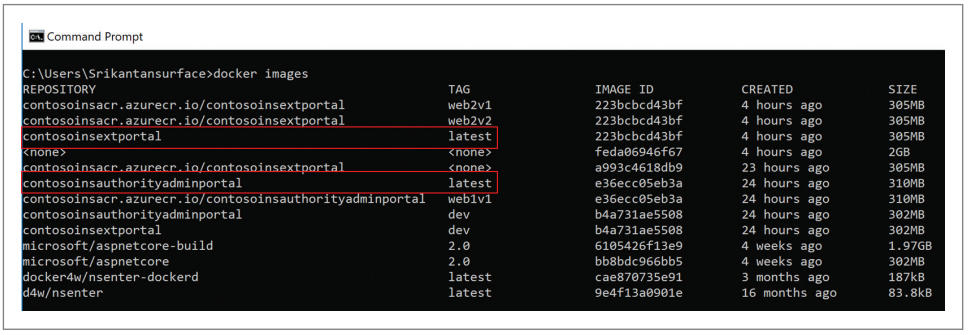


Figure 2 **Docker Images of the Web Applications**

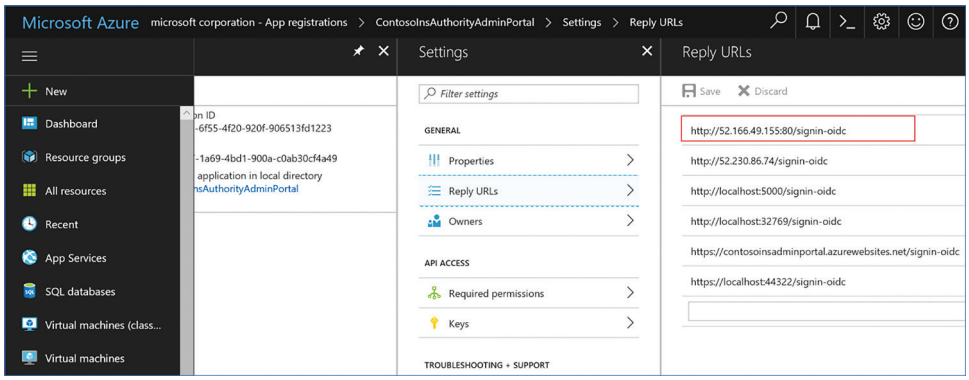


Figure 3 **Configured Reply URLs—Azure AD App Registration**

registration pages in Azure AD for the admin portal and in Azure AD B2C for the consumer portal. **Figure 3** shows the reply URL that's configured for a Web application deployed to ACI or AKS.

> Before you can deploy the application to AKS from your local developer machine, you must have the "kubectl" command tools for Windows installed.

Deploying to Azure Web App for Containers Deploying an application to Azure Web App for Containers is similar to deploying to Azure Container Instances in that there are no virtual machines to provision and no orchestrator for dealing with more complex scenarios. Instead, the containers are deployed to a managed Azure App Service Instance endpoint in Azure.

Figure 4 shows the settings required to create an Azure Web App for Containers instance in the Azure Portal. The URL to the Docker container image uploaded to the Azure container registry is captured as part of the configuration.

Deploying to Web App for Containers provides other handy features that are available with Azure App Service, like the ability to scale out the Web app, configure auto scaling, and use deployment slots for staging and production.

Deploying to Azure Container Services for Kubernetes (AKS) The first step here is creating and configuring an AKS cluster. Before you can deploy the application to AKS from your local developer machine, you must have the "kubectl" command tools for Windows installed.

While the procedure can also be performed from the Azure Portal, the steps I'll describe here are to provision an AKS cluster in Azure using the Azure CLI. Note that at this time, while ACS is generally available across all the Azure regions, AKS, which provides managed Kubernetes clusters with enhanced capabilities not available in ACS, and which is currently in Preview, is available in only a few regions around the world. In this case, I'm creating the cluster in West Europe. Here are the commands I use to do so:

```
az group create --name contosoinsacsrg --location westeurope
az aks create -g contosoinsacsrg --name contosoinsacsportal --generate-
ssh-keys --admin-username <username> --node-count 1
az aks get-credentials -g contosoinsacsrg -n contosoinsacsportal
```

To enable the AKS cluster to access the Azure container registry in order to pull the container images, the service principal of the AKS cluster is given a "Reader" role; that is, it's given read access to the ACR instance hosting the images. Refer to the documentation at bit.ly/2ErvcBq for more details on this. Here are the CLI commands you use to perform these actions:

```
# Get the id of the service principal configured for AKS
az aks show --resource-group contosoinsacsrg --name contosoinsacsportal
--query "servicePrincipalProfile.clientId" --output tsv

# Get the ACR registry resource id
az acr show --name contosoinsacr --resource-group contosoInsAuthRG --query
"id" --output tsv

# Create role assignment
az role assignment create --assignee <ServicePrincipalId> --role Reader
--scope <resourceid>
```

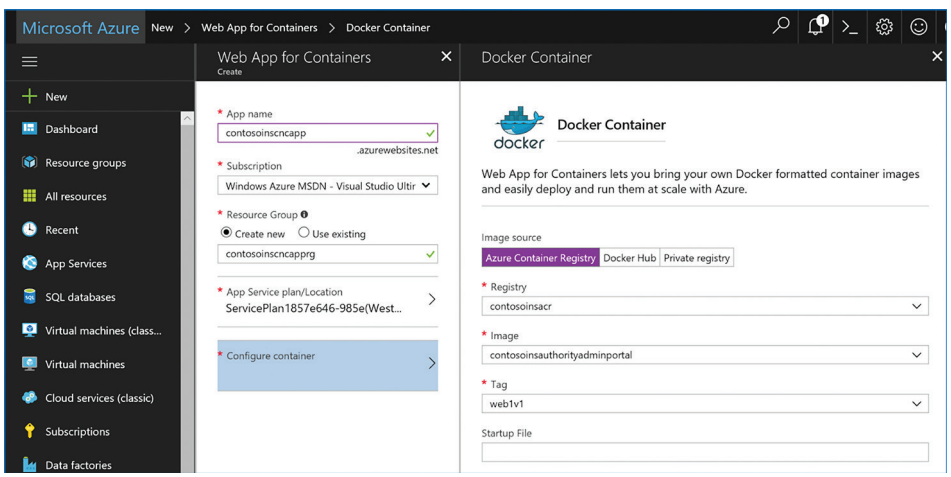


Figure 4 **Azure Web App for Containers**

Now let's deploy the application to the AKS cluster. The steps I'll explain here involve the use of kubectl commands on my development workstation running Windows and Docker.

I'll deploy the application using "YAML" files. (YAML is a data serialization language often used for configuration files.) Figure 5 shows the YAML file used to deploy the admin portal in the solution. As an outcome of this step, the container image of the admin portal is pulled from ACR, provisioned in the AKS cluster and exposed through an Azure load balancer and public IP address.

File Format APIs

ASPOSE
File Format APIs

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

Try risk free - 30 day trial

Download a Free Trial at
https://downloads.aspose.com

Aspose.Words

Create, edit, convert and print Word documents (DOC, DOCX, RTF, etc.) in your .NET, Java and Android applications.

Aspose.Cells

Develop high performance .NET, Java and Android applications to create, edit and convert Excel worksheets (XLS, XLSX, ODS, etc).

Aspose.Pdf

Manipulate PDF file formats (PDF, PDF/A, XPS, etc.) using our native APIs for .NET, Java and Android platforms.

Aspose.Slides

Create, edit and convert PowerPoint presentations (PPT, PPTX, ODP, etc.) in your .NET, Java and Android applications.

Aspose.Email

Create, edit and convert Outlook Email file formats (MSG, PST, EML, etc.) and popular network protocols.

Aspose.BarCode

Generate or recognize barcodes (Code128, PDF417, Postnet, etc.) using our native APIs for .NET and Java.

Aspose.Imaging · Aspose.Tasks · Aspose.OCR · Aspose.Diagram · Aspose.Note · Aspose.HTML

Americas: +1 903 306 1676 EMEA: +44 141 628 8900 Oceania: +61 2 8006 6987
sales@asposeptyltd.com

Figure 5 **The contosoinsportal.yaml File for Deploying the Admin Portal in the AKS Cluster**

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: contosoinsadminv1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: contosoinsadminv1
    spec:
      containers:
      - name: contosoinsadminv1
        image: contosoinsacr.azurecr.io/
contosoinsauthorityadminportal:web1v1
        ports:
        - containerPort: 80
      imagePullSecrets:
      - name: regsecret2
---
apiVersion: v1
kind: Service
metadata:
  name: contosoinsadminv1
spec:
  ports:
  - port: 80
  selector:
    app: contosoinsadminv1
---
apiVersion: v1
kind: Service
metadata:
  name: contosoinsadminv1lb
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: contosoinsadminv1
```

To allow access to the container image in ACR, the credentials must be provided in the YAML file using "pull secrets." The following kubectl command generates a secret file called "regsecret2," which is downloaded to the local machine. This file is referenced in the YAML file used to deploy the application:

```
kubectl create secret docker-registry regsecret2 --docker-server <acr-
login-server> --docker-username <service-principal-ID> --docker-password
<service-principal-password> --docker-email <email-address>
```

Running the next kubectl command deploys the admin portal to the AKS cluster, based on the YAML file passed as an argument.

Browse the Kubernetes dashboard to view the deployed applications, as shown in **Figure 6**. These steps would be repeated for the external portal, after which both applications would be deployed to the AKS cluster, each with its own load-balanced, public-facing endpoints:

```
kubectl create -f contosoinsportal.yaml
az aks browse --resource-group contosoinsacsrg --name contosoinsacsportal
```

Now let's look at the process for deploying application updates. When the next version of your project is available in Visual Studio 2017, the container image in Docker on the local machine is updated. Tag the image in ACR with the new version "web2v2" and push the update to it using the commands that follow. The last command uses kubectl to update the pods in the Kubernetes cluster with the new container image uploaded:

```
docker tag contosoinsextportal:latest contosoinsacr.azurecr.io/
contosoinsextportal:web2v2
docker push contosoinsacr.azurecr.io/contosoinsextportal:web2v2
kubectl set image deployment contosoinsusersv1
contosoinsusersv1=contosoinsacr.azurecr.io/contosoinsextportal:web2v2
```

Figure 7 shows the landing page of the admin portal deployed to AKS.

CI/CD Automation Pipeline for Integration with Azure Container Services

Azure Team Services can be used to implement a continuous integration and deployment pipeline for Docker-enabled applications running in Azure Container Services.

Azure Team Services supports both Team Foundation Server and Git-based repositories for code check-in and version control. For code residing in external repositories like GitHub, a Web hook can be configured in Azure Team Services that would trigger a build process when the code is checked in to the repository.

> To allow access to the container image in ACR, the credentials must be provided in the YAML file using "pull secrets."

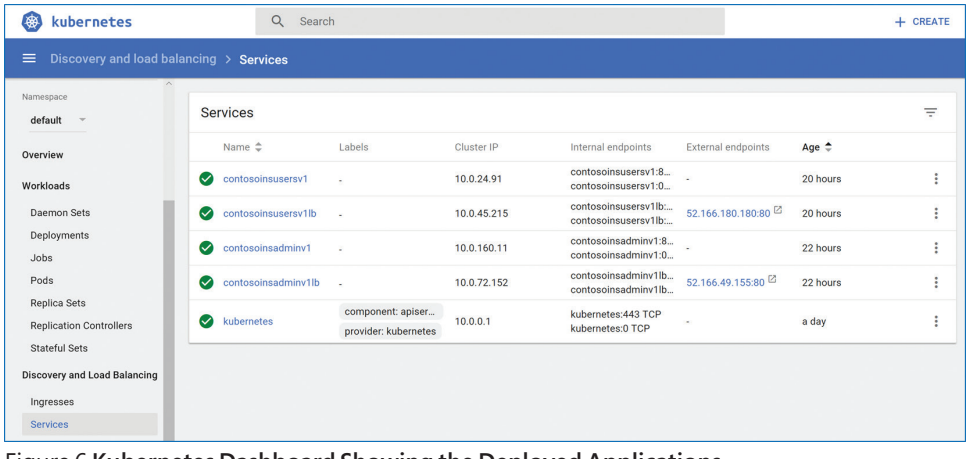


Figure 6 **Kubernetes Dashboard Showing the Deployed Applications**

In the CI pipeline, using the "Docker Image creation" activity would create a new Docker image containing the application updates and push it to Azure Container Registry.

In the CD pipeline, using the "Deploy to Kubernetes" activity—and based on a YAML file—the updated container image could be pushed to an Azure Container Service cluster running Kubernetes.

When using Azure Web App for Containers, the Azure Portal lets you configure the CI/CD pipeline

from the settings blade of the Web App. This configuration handles the generation of a new Docker image of the application when the code is checked in and the build process triggered, and lets the updated Web application be deployed and tested in the staging slot first, prior to pushing the image to the production slot.

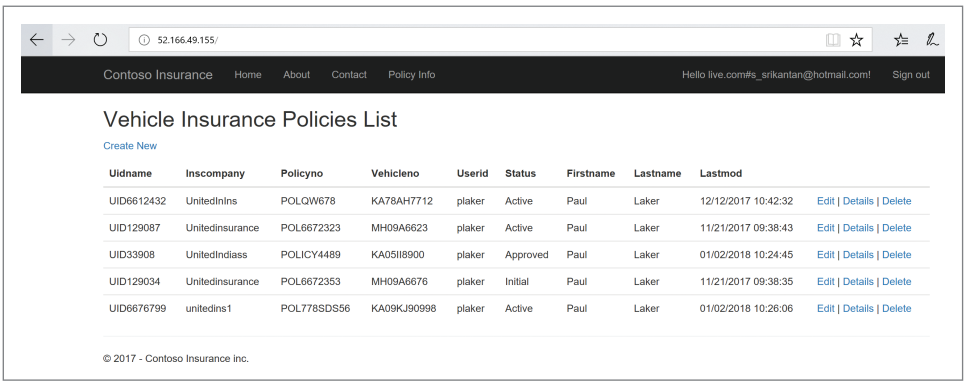


Figure 7 **The Admin Portal Running in AKS**

Enabling SSL on the Web Applications

At this time, with AKS, the Web applications deployed are available using an HTTP endpoint only. To enable the endpoints for HTTPS instead, one of the options available is to deploy Azure Application Gateway, which would expose an HTTPS URL, perform SSL termination, and route the requests to the admin and customer portals.

Azure Application Gateway uses a custom probe to periodically monitor the health of the Web applications in the back-end pool. In order for the health probe to access a page in the Web application to check for application health, minor changes were made to both the portal applications by disabling authentication on the /home page, which doesn't contain business-critical information. This page was then configured in the custom probe definition in Application Gateway to be used for the health check.

At this time, with AKS, the Web applications deployed are available using an HTTP endpoint only.

Alternatively, the Nginx Ingress Controller for Kubernetes supports TLS termination and can be used to SSL-enable access to the Web applications.

Deploying the Solution in Your Subscription

If you'd like to try out this solution on your own, you can download the source files and scripts from the GitHub repository at bit.ly/2DRwdh. You'll need the following software to implement this solution:

- Visual Studio 2017 Preview, Community or Enterprise Edition with Update 3
- An Azure subscription
- A Windows PowerShell script editor
- Azure CLI (bit.ly/2w3J0Ou)
- Docker Community Edition for Windows (dockr.ly/2mnfX7M)
- Visual Studio Tools for Docker (bit.ly/2F6YBRb)
- The Kubectl command tool for Windows (bit.ly/2swZVLJ)

The GitHub repository contains the source files for the admin and customer portals used in the solution. For this article, the single solution that contained the ASP.NET Core 2.0 MVC projects for both Web applications were separated into two individual solution projects so they could be packaged and deployed to containers separately. However, these individual solution files haven't been added to the GitHub repository.

Minor code changes were made in both the projects to suit this article. In the previous article, I used .pfx files in the certificate store on the local machine, which the client embeds in its request to authenticate itself with the Azure Key Vault Service. In this issue, for simplicity, the client secret is used instead, along with the application ID, to authenticate the Web application with Azure Key Vault. The recommended approach is to use certificates in the request for authentication.

To deploy the two solution files to your Azure subscription, refer to the code snippets and reference documentation provided in this article.

Wrapping Up

Azure Key Vault is an effective mechanism that allows businesses to securely manage their sensitive business information using industry standard algorithms and techniques to perform cryptographic operations. Azure provides you with SDKs to consume this service on a variety of platforms in the language of your choice. Without having to make any significant code changes to your application, you could choose to deploy your solutions either as Web applications in Azure App Services or package them as Docker-enabled applications to Azure Container Services or Azure Web Apps for Containers. Visual Studio 2017 tooling for Docker provides a turnkey implementation to Docker-enable a Web application, hiding all the complexity and letting you focus on building business functionality. Azure Container Service gives you access to best-in-class, open source tooling support with Kubernetes for container orchestration. This, combined with the integrated tooling support for continuous integration and deployment with Azure Team Services, ensures organizations have the right tools for agile software delivery across all the phases of a project lifecycle. ■

Srikantan Sankaran *is a principal technical evangelist from the One Commercial Partner team in India, based out of Bangalore. He works with numerous ISVs in India and helps them architect and deploy their solutions on Microsoft Azure. Reach him at sansri@microsoft.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Anil Dwarakanath*

Sensors in Sports: Analyzing Human Movement with AI

Kevin Ashley, Patty Ryan and Olga Vigdorovich

In the future, athletes will likely be able to open their phones and ask a simple question: "What do I need to do to improve my skills?" We're still making early steps in sports AI toward answering that fundamental question, but we hope that the productivity Microsoft tools and research are bringing will one day make this an everyday scenario. With many sports, it's difficult for the human eye to observe all the movements an athlete might make during the course of an activity, but it's possible to record even unobservable data with sensors. And by using machine learning (ML) on this data, the athlete and coach can learn and improve based on precise measurements and analytics. The instrumented athlete is becoming the new competitive advantage.

This article discusses:

- Using sensors to collect athlete data
- Using the Sensor Kit to connect sensors with mobile apps and Azure Cosmos DB
- Processing sensor data with statistical tools, such as R, to extract activity signatures
- Calculating an athlete's load from g-forces using Python

Technologies discussed:

Azure Cosmos DB, R, Python, C#, Sensor Hardware

Code download available at:

bit.ly/2CmQhzq

If the current trend continues, in a few years most sports equipment sold in stores will have a smart sensor embedded. Electronics are becoming smaller, lighter and more flexible, and it's likely we'll see them embedded in fabrics, shoes, skis, tennis racquets and other types of smart gear. You'll be able to determine how to apply technology and skills learned in Internet of Things (IoT), mobile apps, Microsoft Azure and ML to sports.

To make adopting this technology easier, we've created an open source Sensor Kit, with components to process, measure, analyze and improve sensor measurements of athletic performance. Over time, our aim is to evolve this community Sensor Kit together with electronics and sports equipment companies, and sports associations and enthusiasts. The Sensor Kit and all examples for this article are available at bit.ly/2CmQhzq. This includes code samples for R, Python, C#, Xamarin and Azure Cosmos DB. **Figure 1** shows the Winter Sports mobile app, which showcases the use of Sensor Kit and is available for download at winter-sports.co.

The recent dramatic increase in compute power, reliability and affordability of sensor-equipped hardware has made many scenarios newly viable. And advances in applications of AI on sensor signals produced by athletes deliver new ways to understand and improve athletic performance. For example, an athlete's sensor signals provide interpretable "activity signatures," as shown in **Figure 2**, which allow sports analytics to go beyond gross activity tracking and aggregates, and to measure key elements of a skill or activity. From acceleration generated through a specific turn, to directional

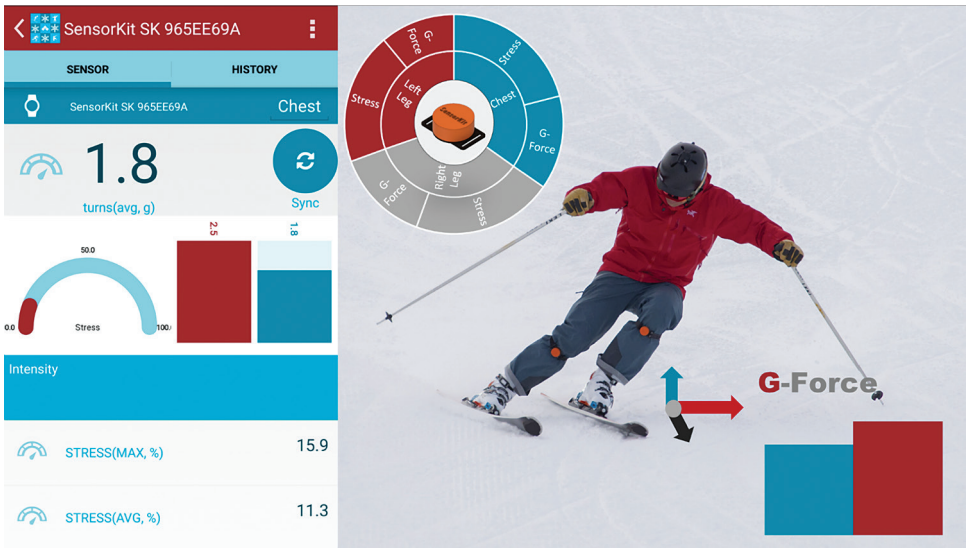


Figure 1 **The Winter Sports Mobile App with Sensor Kit Integration, Illustrating the Forces Impacting a Skier**

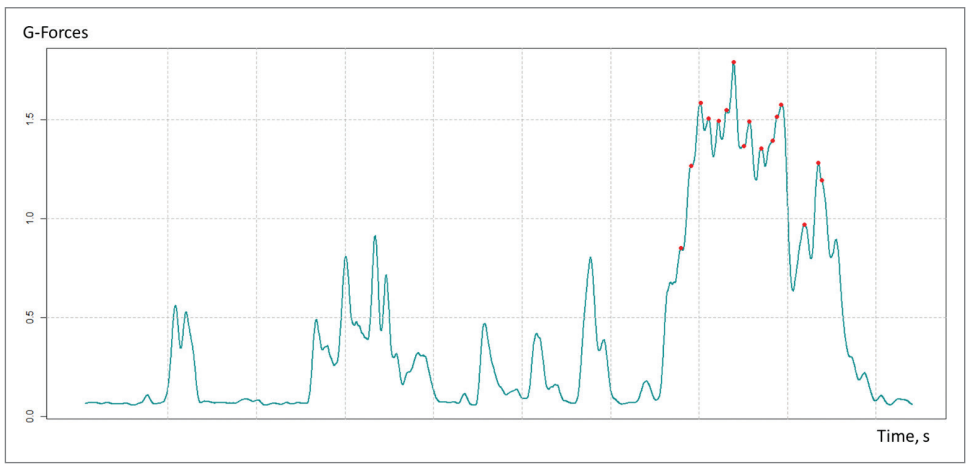


Figure 2 **Activity Signature for Turns in Skiing or Snowboarding**

g-forces engendered during each millisecond, the analytics of sports is being redefined. In this article, we detail how we detect these activity signatures; in this case, the activity in question is turns made while skiing or snowboarding.

Using Azure Artificial Intelligence (AI) in the cloud, we ingest the sensor data, transform it for rich analytics, and tap ML to extract even more useful information for the athlete and coach. ML models make it possible to classify expertise level at each skill execution, and even perhaps predict the progress and future performance of an athlete at an upcoming competitive event. The beauty of shared data standards is this allows athletes to benchmark relative to themselves or their community to understand differences, weak points and advantages. And with new advances in the ability to implement AI at the edge, we can push activity recognition, predictive model scoring and performance measures out to the device, for rapid availability to the athlete on their device or in a mixed reality display. We hope this set of open source resources nurtures and accelerates innovation in the sports community.

Sensor Kit Overview

The Sensor Kit is a set of open source cross-platform data productivity tools, including working code samples for data ingestion, analysis and ML, as well as sensor hardware reference designs. The Kit helps sports scientists, coaches and athletes capture an athlete's movement with millisecond precision, and ML models evaluate the data for movement analysis. The Kit is designed to help equipment manufacturers, data and sports scientists, and coaches interested in modern methods of sports science, including ML and AI. **Figure 3** gives a high-level view of the elements of the Sensor Kit. In the sections that follow, we describe many elements of the Sensor Kit, from hardware, data ingestion and data transformation, to analytics, ML and presentation.

The Kit is designed to allow the sports enthusiast or professional to use parts or all of its components. For example, the analyst may simply want a way to ingest the sensor data signals and transform that data into an analytics-ready format. Data pipeline code samples are available for several typical raw data formats. Or the analyst may want to go further to leverage logic or code to transform the data into consumable analytics reports or predictive ML models. We offer several code sets that recognize specific athletic activities and the performance measures of those activities, for example, turns and the acceleration realized out of the turn. The Sensor Kit SDK is written with cross-platform development in mind, as a cross-platform .NET Standard 2.0 C# library and it compiles with Xamarin, so you can use it in Android, iOS and Windows apps.

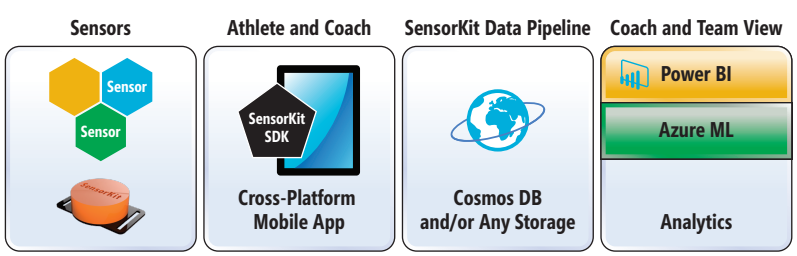


Figure 3 **Sensor Kit for Sports Applications**

Sensor Hardware

Figure 4 shows the Sensor Kit hardware. What is a sensor? Hardware manufacturers usually refer to sensors as single-purpose measuring devices. In this article, a sensor is an inertial measurement unit (IMU) device powerful enough to take and process data from multiple sensory inputs, store the results and transmit them back to the host or a gateway. The sensor system consists of:

Figure 4 **Sensor Kit Hardware**

- An IMU capable of providing 9-DOF (degrees of freedom) accelerometer, gyro and magnetometer data at 40-100Hz.
- Bluetooth Low Energy (BLE) or other means of wireless communication.
- Storage adequate for aggregate or raw sensor data.
- Other sensors, such as those for proximity, location, and so forth.

The Data Model and Pipeline

As the saying goes, a journey of a thousand miles begins with the first step. And so we start with in-depth code examples to ingest, transform and prepare the data for later analysis and presentation. We describe two foundational elements here, designing the data structure or "data model," and designing the pipelining of data from the device to storage, then transforming and presenting the information to the athlete and coach.

Let's start with the data model. The Sensor Kit has three modes of data aggregation—summary, event level and raw logging:

- Summary data is pre-aggregated data on the sensor. In most retail sensor scenarios, this is a relatively small payload that's transferred with each dataset.
- Event-level data is triggered by specific events, such as turns, jumps and so forth. Event-level data may have several hundred or thousand records per session. These events are based on predefined activity signatures derived from the sensor data. They're described in more detail in the turn "Detecting Activity Signatures from Sensor Data" section.
- Raw logging is best for recording data at high frequency, typically the top frequency the sensor can provide—40-100Hz or more.

Depending on the sample rate of the sensor, aggregation of the raw logging data may be necessary to allow collection of data from the device in near-real time. In broader consumer mass market scenarios, producing, storing and transmitting so much granular data can be excessive, so we defined a standard mode in the Sensor Kit to transmit slightly aggregated data to reduce data throughput requirements. If you need all the detailed data, the Sensor Kit allows you to enable verbose logging using the SetLogging(true) call.

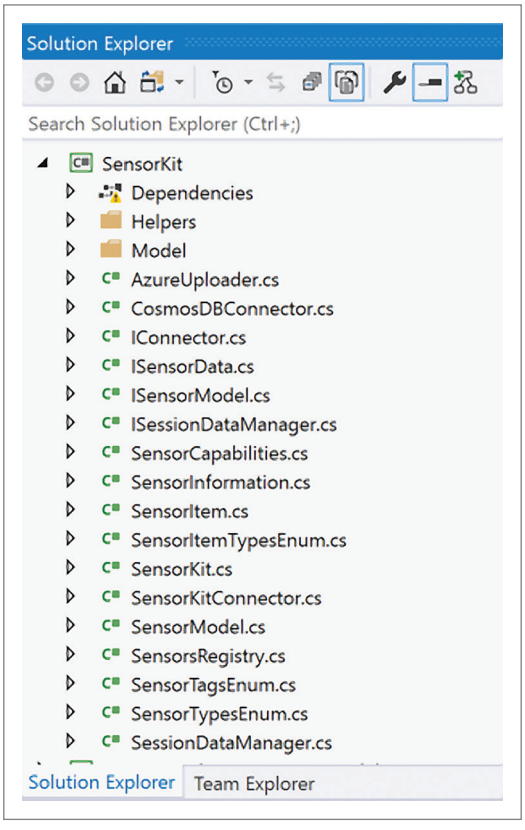


Figure 5 **The Sensor Kit Library in Visual Studio**

Connecting to Sensors

Let's start the description of data pipelining from the sensor itself. Initializing the Sensor Kit is easy, by simply calling the Init method:

```
SensorKit.Instance.Init();
```

The Sensor Kit works in both pull and push modes; with pull mode the app needs to explicitly call the Sync method on a sensor, while push mode automatically registers for sensor notification. To enable push updates, you can call SetAutoUpdates(true). To subscribe to sensor updates, use the Subscribe method:

```
await Task.Run(async () =>
{
  await sensor.Instance.Subscribe();
});
```

The Sensor Kit consumes data from sensors, provides methods for time synchronization and sends the data to the cloud. Time synchronization is important, especially when athletes can have multiple sensors attached, and the Sensor Kit automatically resolves the timestamp on hardware devices to the time on the host device with the Sensor Kit-enabled app. The method to store data in the cloud is up to the app developer; the library provides a Cosmos DB connector and Azure containers for convenience, as shown in **Figure 5**.

The Sensor Kit delivers some transformed data events and aggregates from the device itself. The following list of item schema describes the taxonomy of these data elements:

- SensorItem: Single-sensor data item; items can be of any duration or granularity
- SensorTurnData: Aggregated data for turns
- SensorAirData: Aggregated data for jumps
- SensorSummaryData: Summary data aggregated per sensor
- SensorRawData: High-frequency raw data (for example, 100Hz)
- UserData*: User-level information (optional, app specific)
- TeamData*: Team-level data for teams of athletes (optional, app specific)

Storing Sensor Data in Cosmos DB

Of course, there are many options when it comes to loading data into the cloud. Cosmos DB is great for IoT and telemetry data, as shown in **Figure 6**, and provides multiple APIs for loading and querying data, as well as scalability and global distribution.

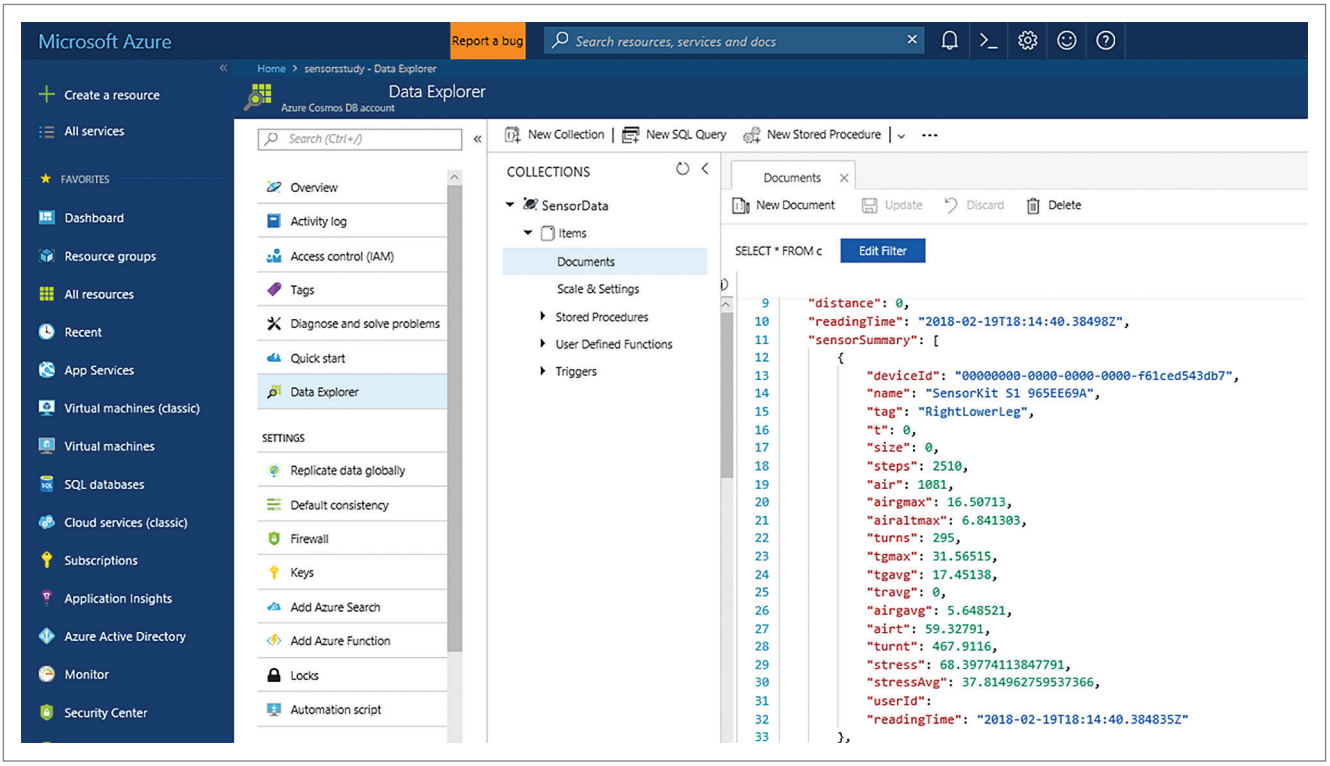


Figure 6 **Cosmos DB with Sensor Kit Documents**

The Sensor Kit includes a Cosmos DB connector and Azure Functions for storage containers, which you can find at bit.ly/2GEB5Mk. You can easily update data from the sensors with the Sensor Kit connected to Cosmos DB by using the following method:

```
await SensorKit.AzureConnectorInstance.
InsertUserSensorDataAsync(userSensorData);
```

Athletes can have multiple sensors, and the Sensor Kit aggregates sensor data at the athlete level and updates Cosmos DB with new data. Once the data is in Cosmos DB, it's easy to query that data via multiple interfaces, both SQL and non-SQL. For example, you can use Microsoft Power BI to create a specialized coach's view of the data from the sensors of every athlete on the team, or use the mobile app to present the data. This following query returns summary data from each sensor found via the Sensor Kit, as shown in **Figure 7**:

```
SELECT * FROM Items.sensorSummary c
```

Once the data is uploaded to Azure, you can use ML to train your models or process the data.

Now that you know how the data gets into the cloud, let's focus on the logical part of sensor data analysis and, more specifically, the analysis for the sports domain. From working with coaches, we determined that they're interested in detecting events, such as turns, and load from g-forces experienced by athletes during runs. Let's see how we can parse collected data to detect skiing turns.

Detecting Activity Signatures from Sensor Data

The main trajectory of the skier is aligned with movement of his center of mass, so we placed a sensor in the middle of the pelvis, inside the pocket in the ski jacket. We received data from both an accelerometer and a gyroscope. Using information from the sensor allowed us to analyze the athlete's movement and define an activity signature.

Our data consists of accelerometer and gyro values stored in a sample text file (bit.ly/2GJkk2w). Our 9-DOF sensors give us a 3D acceleration and angular velocity vectors from the accelerometer and gyroscope, respectively, sampled at approximately 100Hz. To analyze the data we load it into RStudio. You can use our parsing code, which is available at bit.ly/2GLc5mG, and load our sample files.

> Working with 9-DOF sensors requires thorough calibration of the sensor, which is a very tedious procedure.

Working with 9-DOF sensors requires thorough calibration of the sensor, which is a very tedious procedure. Our goal in this case was simply to calculate the number of turns, which doesn't require performing a precise movement analysis along a specific axis. For this article, and to simplify our calculations, we're using the magnitude of the acceleration and angular velocity.

Because our experiment involves detecting turns, we're only somewhat interested when the athlete is moving. When the athlete stands still, the accelerometer sensor shows almost a flat line, while the angular velocity might still be changing. When the actual movement starts, the amplitude of the acceleration and gyro values changes rapidly, so the standard deviation increases. As **Figure 8** shows, we can define an activity starting point at the beginning of the data where accelerometer values exceed a fixed threshold. The

TEXT CONTROL

TX Text Control X15

Automate your reports and create beautiful documents in Windows Forms, WPF, ASP.NET and Cloud applications.

Text Control Reporting combines powerful reporting features with an easy-to-use, MS Word compatible word processor. Users create documents and templates using ordinary Microsoft Word skills.

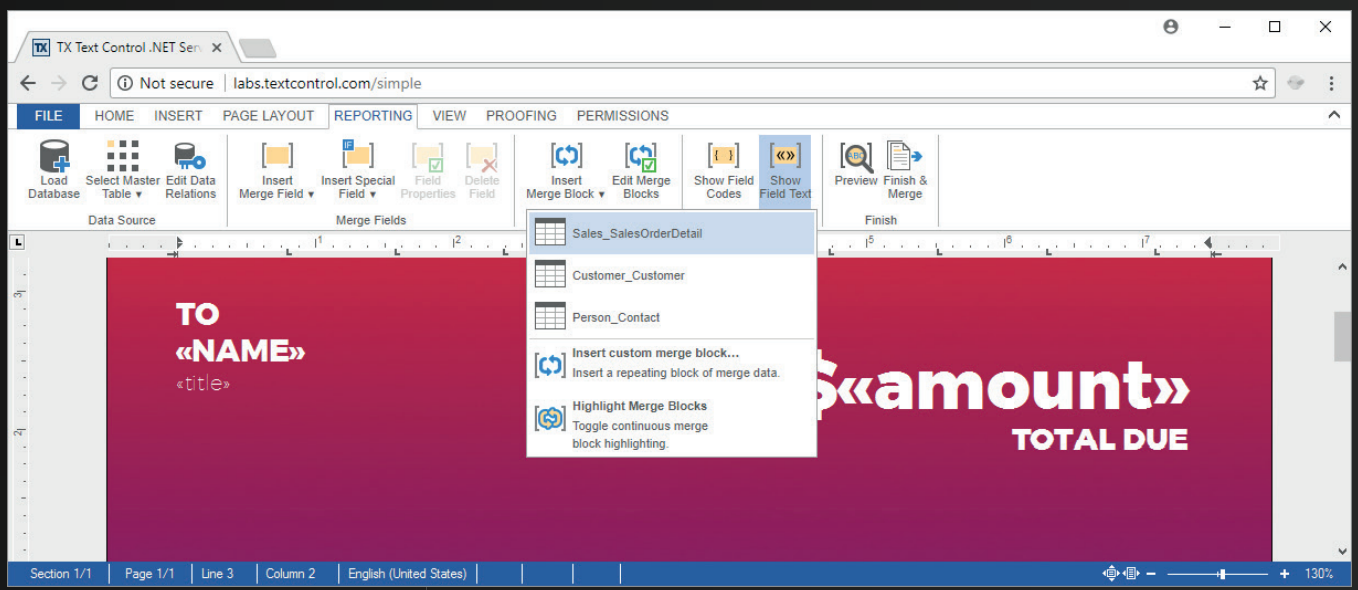
Download a free trial at
www.textcontrol.com

WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio Partner

TX Text Control .NET Server for ASP.NET

Complete reporting and word processing for ASP.NET Web Forms and MVC



- Give your users a WYSIWYG, MS Word compatible, HTML5-based, cross-browser editor to create powerful reporting templates and documents anywhere.
- Text Control Reporting combines the power of a reporting tool and an easy-to-use WYSIWYG word processor - fully programmable and embeddable in your application.
- Replacing MS Office Automation is one of the most typical use cases. Automate, edit and create documents with Text Control UI and non-UI components.

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

TEXT CONTROL

Documents | Query 1 ×

Execute Query

```
1 SELECT * FROM Items.sensorSummary c
```

Results: 1 - 11 | Request Charge: 8.3 RUs | Next →


```
[
    [
        {
            "deviceId": "00000000-0000-0000-0000-f61ced543db7",
            "name": "SensorKit S1 965EE69A",
            "tag": "RightLowerLeg",
            "t": 0,
            "size": 0,
            "steps": 5496,
            "air": 238,
            "airgmax": 33.87532,
            "airaltmax": 0.483613,
            "turns": 501,
            "tgmax": 33.92054,
            "tgavg": 15.62689,
            "travg": 0,
            "airgavg": 7.11544,
            "airt": 11.505,
            "turnt": 1049.898,
            "stress": 86.32614256972994,
            "stressAvg": 39.76968332642956,
            "userId":                                    ,
            "readingTime": "2018-02-15T23:41:43.909428Z"
        },
        {
            "deviceId": "00000000-0000-0000-0000-c974a2c13fa7",
            "name": "SensorKit S1 964C478B",
            "tag": "LeftLowerLeg",
            "t": 0,
            "size": 0,
            "steps": 5785,
            "air": 329
```


Figure 7 **Cosmos DB Query Results for Sensor Kit Summary Data**

ending point will be toward the end of the data where accelerometer goes below that threshold.

It's well known that accelerometer sensor data is very noisy, and using a moving average calculation yields a smoother signal. Let's define a point in time when the standard deviation exceeds the threshold as a percentage of the average value of the signal as a starting point:

```
a_smooth1 <- SlidingAvg(a,lag,threshold,influence)
st<-which(a_smooth1$std > thresholdPct* a_smooth1$avg)
startPos<-min(st)
endPos<-max(st)
```

One way to detect peaks in data is to use the property that the magnitude of a peak must be greater than its immediate neighbors. Here's how we calculate the magnitude of the accelerometer and gyro values:

```
lag       <- 30
threshold <- 1.5
influence <- 0.5
res<-SmoothAndFindPeaks(df$magnitudeW,df$magnitudeA,lag,threshold,influence)
print (paste("Total peaks in A = ", length(res$pks_a), " peaks in W = ",
length(res$pks_w) ))
```

which results in:

```
> res<-SmoothAndFindPeaks(df$magnitudeW,df$magnitudeA,lag,threshold,influence)
[1] "Calculation for activity start = 3396 end = 4239"
> print (paste("Total peaks in A = ", length(res$pks_a), " peaks in W = ",
length(res$pks_w) ))
[1] "Total peaks in A = 29 peaks in W = 22"
```

These results are illustrated in **Figure 9**.

> One way to detect peaks in data is to use the property that the magnitude of a peak must be greater than its immediate neighbors.

Observation of the movement of athletes while turning shows that every time during the transition phase of the turn, the magnitude goes to a lower value, and at the apex of the turn the magnitude gets to its maximum. This means that if we count peaks in the "moving" segment of the data we'll get our desired number. The gyro data is also much cleaner and we'll use angular velocity magnitude to calculate peaks to determine the turns. Afterward, we need to get rid of peaks that are too close to each other.

To get low noise data that's stable over time, we could use a complementary filter that combines the long-term stability of the accelerator with the short-term accuracy of the gyroscope.

Now that we have R code for turn detection, we can build an ML training model in a process similar to the one that's described in a Machine Learning Blog post at bit.ly/2EPosSa.

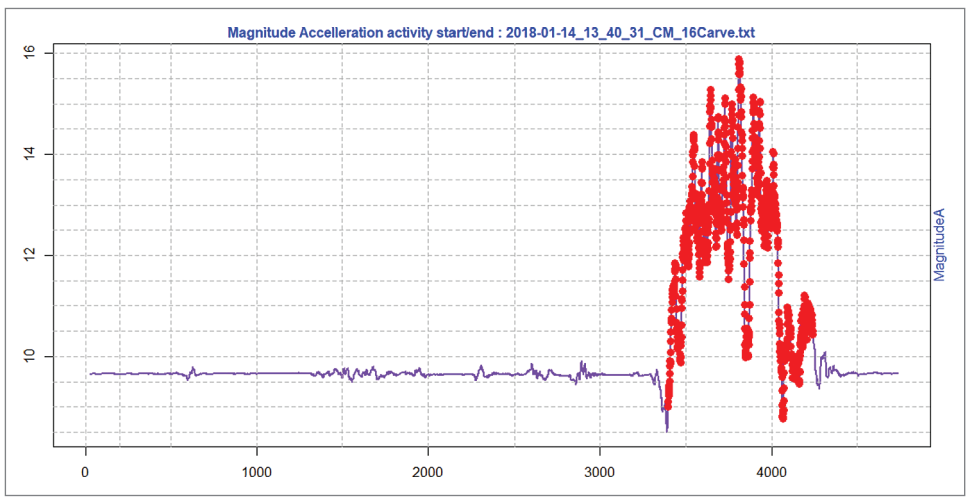


Figure 8 **The Start and End of Acceleration Activity**

Measuring Athlete Load with G-Forces

Athletic load in a workout goes beyond simply the aggregate measures of distance traveled or total activities completed. A key aspect of understanding an athlete's stress and workout quality involves measuring the load on the athlete. There

Learn, Explore, Use

Your Destination for Data Cleansing & Enrichment APIs

Your centralized portal to discover our tools, code snippets and examples.

RAPID APPLICATION DEVELOPMENT

Convenient access to Melissa APIs to solve problems with ease and scalability.

REAL-TIME & BATCH PROCESSING

Ideal for web forms and call center applications, plus batch processing for database cleanup.

TRY OR BUY

Easy payment options to free funds for core business operations.

FLEXIBLE CLOUD APIS

Supports REST, JSON, XML and SOAP for easy integration into your application.

Turn Data into Success – Start Developing Today!

Melissa.com/developer

1-800-MELISSA

melissa

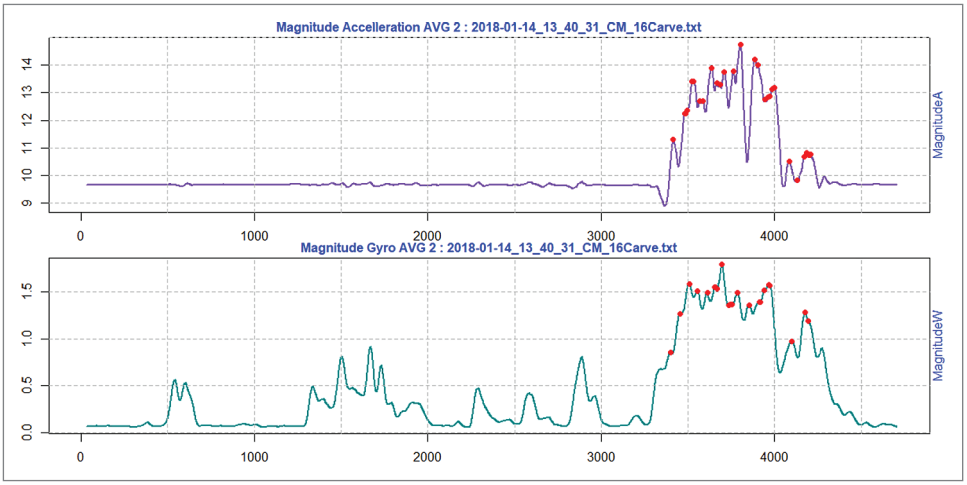


Figure 9 **Finding Peaks in Accelerometer and Gyro Values**

are several components to athletic load. One of the most important involves the g-forces generated and experienced by the athlete, which creates forces they need to control over the course of an activity like a ski turn, as well as jerks and snaps generated from the terrain or their motion that they must accommodate.

Humans have limits related to g-forces they can tolerate. These limits depend on the amount of time the stress is experienced and whether it's a low intensity over a long duration or a high intensity for short durations. And these limits depend as well on the direction in which that g-force is felt. For example, humans are much more capable of tolerating high vertical-direction g-force (the "z" access) rather than a high lateral-direction g-force (the "y" access) with its stress on the neck, back and joints. Fortunately, there have been extensive field studies of g-force effects and limits from the aviation industry and from NASA that we can leverage to measure and characterize g-force and g-force tolerances. Let's delve into g-forces on the human body, how we measure them, and how to represent the load they create on the athlete.

G-Force Calculations from Acceleration Sensor Measures Calculating directionless g-force on the human body with our sensor acceleration measures is as follows using the Pythagorean theorem:

Directionless G-Force = Math.sqrt(AccelerationX^2 + AccelerationY^2 + AccelerationZ^2)

G-force in a given direction is calculated by dividing acceleration by 9.81, assuming acceleration is measured in meters per second squared. For example, Y-direction g-force is calculated as follows:

Y G-Force = AccelerationY / 9.81

Of course, an important component of the g-force load on the athlete is the time period during which it's felt.

G-Force Maximum Tolerances by Duration For athletic activity, a useful comparison is the g-force maximums that the human body can tolerate in each direction. And while these g-forces reflect maximums for the center of gravity, rather than for a particular joint, they offer a useful way to express the g-forces experienced by an athlete as a percentage of these maximums. Given that we're able to measure g-force durations at the sample rate of the sensor, in anywhere from 10 samples per second (10hz) to 100 samples per second (100hz), we can characterize the athlete's g-force load as a percentage of maximum for any given duration.

Python Code to Calculate G-Forces As noted earlier, the formula just requires the Acceleration X, Y and Z measures from the sensor. And you need to understand the scale in which it's measured, typically either in feet per second squared or in meters per second squared. From these elements, we can calculate g-force in a specific direction, as well as directionless g-force being experienced overall by the athlete's body. Thanks to our partner XSens, we use high-precision sensors to record data movements at 100 samples per second (100hz) and load them into Azure.

> A key aspect of understanding an athlete's stress and workout quality involves measuring the load on the athlete.

Figure 10 shows a capture from the sensors mapped to a 3D animation. As you can see, the skier is experiencing g-force in the Y-direction, as well as in the X-direction as he travels down the hill. Calculating these forces, in combination with the acceleration achieved coming out of that turn, allows athletes to better understand their performance at navigating that g-force through the turn.

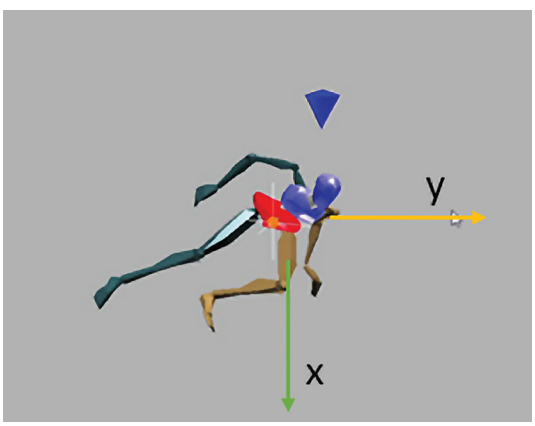


Figure 10 **G-Force Load Is Different Along Different Axes, as Visualized by XSens Software**

Figure 11 presents the Python code for calculating g-force.

You'll find the script to calculate g-force, as well as additional calculations in the script including g-force relative to maximums, on our GitHub rep at bit.ly/2BPS6nA. To read more about the physics of g-forces, and their impact on humans, take a look at the "Beyond Velocity and Acceleration: Jerk, Snap and Higher Derivatives" article from the *European Journal of Physics* at bit.ly/2FvLkTD, which describes g-forces relative to the context of the roller-coaster

Figure 11 **Python Code for Calculating G-Force**

```
########################
# G-Force, Using Python 3.5+
########################

#Directionless g-force (gg = Math.sqrt(Accx * Accx + Accy * Accy + Accz *
Accz))  <pythagorean theorem>
#Assuming accelerometer is in meters per second squared, gforce
measurement by dividing by 9.81
#1 acceleration of gravity [g] = 9.80664999999998 meter/second² [m/s²]
#our data is in feet per second so we use the converstion 1 ft/s2 = 0.3048 m/s2

#Set Conversion Metrics
G_conversion = (9.80664999999998)
MperS_conversion = (.3048)  #from above

#Replace na's with zeroes to avoid math errors
dataset=dataset.fillna(0)

#Using acceleration measures from 'dataset' dataframe, we convert to
meters per second squared
#In our case our acceleration variables are labeled AccX, AccY and AccZ
#In our case, acceleration was in feet per second, so we needed to apply
a conversion.
dataset["AccX_mtrpersecsqrd"] = dataset["AccX"]/MperS_conversion]
dataset["AccY_mtrpersecsqrd"] = dataset["AccY"]/MperS_conversion]
dataset["AccZ_mtrpersecsqrd"] = dataset["AccZ"]/MperS_conversion]

#Generate Directionless G-Force measure, call it 'DirectionlessGG'
dataset["DirectionlessGG"] = ((dataset["AccX"]*dataset["AccX"])+(dataset[
"AccY"]*dataset["AccY"])+dataset["AccZ"]*dataset["AccZ"])).astype(float)
dataset["DirectionlessGG"] = np.sqrt(dataset["DirectionlessGG"])#.astype(float)

#Generate Direction Specific G-Force, call them 'X_GG', 'Y_GG' and 'Z_GG'.
dataset["X_GG"] = dataset["AccX"]/MperS_converstion/G_conversion
dataset["Y_GG"] = dataset["AccY"]/MperS_converstion/G_conversion
dataset["Z_GG"] = dataset["AccZ"]/MperS_converstion/G_conversion
```

experience. And for an understanding of human tolerances and limits of g-forces, refer to the Wikipedia article at bit.ly/2EPQDjE, as well as to the NASA collection of research at go.nasa.gov/2oyS9fj. More information on calculating g-force is available at bit.ly/2Fzxpfa.

Wrapping Up

We used custom-built and partner-made sensors to collect athlete data, and illustrated the use of our open source Sensor Kit that connects sensors with mobile apps and Azure Cosmos DB. We explained how to process that data with statistical tools, such as R, to extract "activity signatures" that describe the turns athletes make while skiing. Finally, we explained how to use data from sensors to calculate an athlete's load from g-forces using Python. ■

Kevin Ashley *is an architect evangelist at Microsoft. He's coauthor of "Professional Windows 8 Programming" (Wrox, 2012) and a developer of top apps and games, most notably Active Fitness (*activefitness.co*). He often presents on technology at various events, industry shows and webcasts. In his role, he works with start-ups and partners, advising on software design, business and technology strategy, architecture, and development. Follow him on Twitter: @kashleytwit.*

Olga Vigdorovich *is a database administrator, data scientist and an avid skier. She built the data model and back end for scalable cloud platforms based on Microsoft Azure, including Winter Sports, for Active Fitness at Summit Data Corp.*

Patty Ryan *is an applied data scientist for Microsoft. She codes with its partners and customers to tackle tough problems using machine learning approaches, with sensor, text and vision data. Follow her on Twitter: @singingdata.*

Thanks to the following Microsoft technical expert who reviewed this article: *Mona Soliman Habib*

Using Fluent: Swipe Gesture Action and Connected Animations

Lucas Haines

Application developers have more and more device types to deal with, from ubiquitous smartphones and traditional desktop and laptop PCs, to large-format touchscreen displays and even emerging virtual and augmented reality platforms like HoloLens. The Fluent Design System is specifically designed and continually evolving to make it is easier for developers to integrate modern features for the mixed device ecosystem. The Fluent features added to the Windows 10 platform in the Fall Creators Update are a powerful case in point, enabling applications to feel natural, no matter on what device they run.

Fluent UI features like light, blur and depth combine to enable compelling and intuitive experiences that allow users to do more. But until recently, challenges remained for users trying get the most out of their devices. With the Windows 10 Fall Creators Update, Fluent adds two features that solve both problems today: Swipe Gesture Actions and Connected Animations.

Swipe Gesture Actions

It comes as no surprise that sales of touchscreen-equipped PCs continue to grow. The question for developers is how to help users do more by enabling interaction models that work both for touchscreen PCs and classic keyboard and mouse.

The answer comes in the form of on-object commanding, which has been around for years via either a right-click or a tap and hold. These tried and true methods are well understood, but with a bit of work you can build in accelerators that leverage the capabilities of the device ecosystem to make them better.

Enter gesture actions, which are great accelerators that work for both touch and keyboard users. Gesture actions are a handy accelerator for contextual commands typically hidden behind a right-click menu or on a separate commanding bar. Other examples of gesture actions include hover buttons, swipe, and keyboard accelerators.

Swipe for commanding isn't a new concept. The interaction pattern exists on both Google Material Design and Apple iOS. It's useful for quickly triaging items in a list—for example, deleting photos or e-mails in a scroll—or for revealing the most-used commands in a scenario. The XAML Swipe Control makes it very easy to add this great functionality to your Universal Windows Platform (UWP) apps.

The Swipe control contains two behaviors, Reveal mode and Execute mode:

Reveal mode can display one or more commands associated with a selected item so that a user can select from among them. For instance, if you have a list of messages and you want to quickly enable forwarding, creating a follow up reminder, or scheduling a meeting, reveal mode makes this possible without forcing the user to navigate screens. This can be very powerful for the "on the go" user.

Execute mode completes the command with a simple gesture. The most common and easiest example would be delete. I no longer

This article discusses:

- Swipe gesture actions in UWP apps
- Connected animations in UWP apps

Technologies discussed:

Windows 10, Universal Windows Platform, XAML, C#, Visual Studio

need this message and want to forget it ever existed. One simple gesture (like a swipe left) and it's off to the recycling bin. Execute mode doesn't have to be destructive. Another great example is swipe to save. Say in one column you have a list of customers and when you select one a detail pane appears to the right. After the user makes changes to the form or customer record, he or she simply swipes the item to save the changes. Simple and fast.

However, before I can create the swipe behavior on my list, I like to set up the SymbolIconSource for my application. This provides a really easy way to access icons for your application. Adding the symbols to page.resources allows me to easily add icons where needed later in the code. Regardless of implementing swipe, this is great at making code more readable and taking advantage of IntelliSense in Visual Studio. Here's the code:

```
<Page.Resources>
  <SymbolIconSource x:Key="ReplyIcon" Symbol="MailReplyAll" />
  <SymbolIconSource x:Key="PinIcon" Symbol="Pin" />
  <SymbolIconSource x:Key="DeleteIcon" Symbol="Delete" />
</Page.Resources>
```

With the resources loaded, I can begin to implement the swipe gesture API on an element. List views are an extremely common element to implement swipe gestures, so that's what I'll work with, as shown in the code in **Figure 1**.

Notice in **Figure 1** that I'm using the LeftItems and RightItems property in the SwipeControl API. LeftItems determines what happens when the user swipes left (in this case, RevealOptions) and places the command buttons to the right side.

The directional item references a Static Resource, which is defined in SwipeItems placed in page resources with my icons source. This is where I set the behavior mode to either Reveal or Execute, like so:

```
<Page.Resources>
  <SwipeItems x:Key="RevealOptions" Mode="Reveal">
    <SwipeItem Text="Reply" IconSource="{StaticResource ReplyIcon}"
      Foreground="White"/>
    <SwipeItem Text="Pin" IconSource="{StaticResource PinIcon}"
      Foreground="White" />
  </SwipeItems>
  <SwipeItems x:Key="ExecuteOptions" Mode="Execute">
    <SwipeItem Text="Delete" IconSource="{StaticResource DeleteIcon}"
      Invoked="SwipeItem_Invoked"
      Background="Red" Foreground="White" />
  </SwipeItems>
</Page.Resources>
```

Now that I have the UI created and the interaction working, I can use the Invoke property to set up an event handler and process the action. Processing the command when a swipe is performed is done on the SwipeItems declaration in page.resources by setting the Invoked property. Here's the code for that:

```
<SwipeItem Text="Delete" IconSource="{StaticResource DeleteIcon}"
  Invoked="DeleteItem_Invoked"
  Background="Red" Foreground="White" />
```

Once in the codebehind, you can add your code for handling the command here:

```
private void DeleteItem_Invoked(SwipeItem sender, SwipeItemInvokedEventArgs args)
{
  int index = myListView.Items.IndexOf(args.SwipeControl.DataContext);
  myListView.Items.RemoveAt(index);
}
```

Even though the origin of swipe is rooted in touch devices, it works great with track pads and pen. I often find that when I'm working with a device that isn't touch-enabled that I check for swipe gestures with the track pad. I'm always delighted when I find that it works. By default, Fluent Swipe Gesture actions work with track pads, giving users more ways to quickly complete their tasks and move on. It's a perfect example of implementing for one input method and benefiting all users.

When thinking of adding swipe to your application, consider if the gesture will conflict with other behaviors in the application. Do you have a FlipView, Hub or Pivot as the main content area in your application? That's probably not the best place to implement swipe because the interaction patterns are very similar and getting the right balance for users can be difficult.

Swipe is great when the same action needs to be repeated multiple times, but remember that the result of the action needs to be consistent. I can't think of a worse scenario than having a user who understands that swiping left will favorite an item, only to find on the next screen that the same gesture deletes it. Finally a tip: In my experience, swipe for touch or mouse users is best if the item being swiped is at least 300 pixels wide.

Connected Animations

Today, many apps and Web sites have "jerky" or non sequitur transitions. When an action is taken that requires page navigation, the user is only presented with the new page. There are few—if any—visual cues to provide context or guidance about what to do, where to go or what is new. With connected animations, you can preserve context for the user and increase engagement, such as by animating the clicked item from a list to the target location of the details page. Or you can keep the user's Avatar visible while they navigate through an application. It's all about maintaining context and focus.

Adding in connected animations is an incremental process. If you want to implement it for all your page transitions, please don't let me stop you. But we all have a backlog of work to burn through, so it may be best to be selective and tackle the effort over time. There are two steps required to implement connected animations:

1. Prepare an animation object on the source page.
2. Start the animation on the destination page.

Preparing the animation on the source page allows you to set the source object that's to be animated across the pages. The same image source should be set on the destination page, though you can use a lower-resolution image for the target page. This can reduce the memory footprint of your application. The connected animation will then crossfade between the two images.

The general rule of thumb with connected animations is that animation should start approximately 250 milliseconds between

Figure 1 **Implementing Swipe on a ListView**

```
<ListView x:Name="MainList" Width="400" Height="500">
  <ListView.ItemTemplate>
    <DataTemplate x:DataType="x:String">
      <SwipeControl x:Name="LVSwipeContainer"
                    LeftItems="{StaticResource RevealOptions}"
                    RightItems="{StaticResource ExecuteDelete}">
        <StackPanel Orientation="Vertical" Margin="5">
          <TextBlock Text="{x:Bind}" FontSize="18" />
          <StackPanel Orientation="Horizontal">
            <TextBlock Text="Data Template Font" FontSize="12" />
          </StackPanel>
        </StackPanel>
      </SwipeControl>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

the two steps or else the source element may hang in the view and end up looking weird. With the implicit animation engine shipped in Windows 10, if you prepare an animation but don't start it within three seconds, the system will dispose of it.

For this scenario I'm going to use ListView with some text and images, using the code provided in **Figure 2**. Both Listview and Gridview have two methods added specifically for connected animations: PrepareConnectedAnimations and TryStartConnectedAnimationAsync.

> With connected animations, you can preserve context for the user and increase engagement, such as by animating the clicked item from a list to the target location of the details page.

The key element in this list is the x:Name of the image in the DataTemplate. This is the name that I'll use when I create the ConnectedAnimation and prepare it for the destination page. When an item in the collection is clicked it will navigate to the new page. I'll prepare the connected animation during the click method. The event handler set on the list Collection_ItemClicked is where I prepare the ConnectedAnimation before navigation. The PrepareConnectedAnimation method expects a unique key, the item, and the name of the element to be animated. In the method I've named my animation "ca1," which will reference on the destination page when the animation is started, as shown in **Figure 3**.

SuppressNavigationTransitionInfo blocks the default page transition animation from playing, and helps prevent it from interfering with the connected animation. Using the OnNavigatedTo method for the destination page, I create a ConnectedAnimation and pass in the unique key I created on the source page ("ca1"). Then I call TryStart and pass in the name of the XAML image element I want to animate to, using this XAML code:

```
<Image x:Name="detailedIamge" MaxHeight="400" Source="{x:Bind ImageLocation}" />
```

And this C# code:

```
ConnectedAnimation imageAnimation =
  ConnectedAnimationService.GetForCurrentView().GetAnimation("ca1");

if (imageAnimation != null)
{
  imageAnimation.TryStart(detailedIamge);
}
```

This creates the one-way connected animation from the listview to the destination page. I still need to create another connected animation to handle the back navigation scenario. I prepare the connected animation in the OnNavigateFrom override and give it a unique key, "ca2," as shown in the following code:

```
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
base.OnNavigatedFrom(e);

ConnectedAnimationService.GetForCurrentView().PrepareToAnimate("ca2", detailedIamge);
        }
```

The ca2 animation is started using the collections loaded method declared in the listview template, like so:

```
private async void Collection_Loaded(object sender, RoutedEventArgs e)
{
if (_storeditem != null)
{
ConnectedAnimation backAnimation =
  ConnectedAnimationService.GetForCurrentView().GetAnimation("ca2");

if (backAnimation != null)
{
await collection.TryStartConnectedAnimationAsync(backAnimation, _
  storeditem, "ConnectedElement");
}
}
}
```

Figure 2 **Setting Up ListView Collection to Animate**

```
<ListView x:Name="Collection"
                ItemClick="Collection_ItemClick"
                ItemsSource="{Binding Source={StaticResource ItemsViewSource}}"
                IsItemClickEnabled="True"
                SelectionMode="None"
                Loaded="Collection_Loaded"
                Grid.Row="1">
  <ListView.ItemTemplate>
    <DataTemplate x:DataType="local:CustomDataObject">
      <Grid Margin="0,12,0,12">
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto" MinWidth="150" />
          <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <!-- image to be animated-->
        <Image x:Name="ConnectedElement" Source="{x:Bind ImageLocation}"
          MaxHeight="100"
          Stretch="Fill" />

        <StackPanel Margin="12,0,0,0" Grid.Column="1" >
          <TextBlock Text="{x:Bind Title}" HorizontalAlignment="Left"
            Margin="0,0,0,6" />
          <StackPanel Orientation="Horizontal" >
            <TextBlock Text="{x:Bind Popularfor}" />
          </StackPanel>
        </StackPanel>
      </Grid>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

Figure 3 **PrepareConnectedAnimation Method**

```
private void Collection_ItemClick(object sender, ItemClickEventArgs e)
{
  var container = collection.ContainerFromItem(e.ClickedItem) as ListViewItem;

  if (container != null)
  {
    _storeditem = container.Content as CustomDataObject;

  var animation = collection.PrepareConnectedAnimation("ca1", _
    storeditem, "ConnectedElement");
  }

  Frame.Navigate(typeof(DestinationPage), _storeditem);

}
```

I'm using the async TryStart method called on the listview, to ensure that the listview content is rehydrated before the animation starts.

Typically with lists or dense views of data you have lots of secondary data associated with the item—for example, mail subject, sender, date/time and the like. We can aid the user even further by animating these elements into view. This requires using CoordinatedAnimation, which I can do by using the two-parameter overload for TryStart.

First, I need to create an element on the destination page to show the appropriate content. I'm using a stackpanel with a text block inside and I've named the stackpanel CoordinatedPanel, as shown here:

```
<StackPanel x:Name="CoordinatedPanel" Grid.Column="1"
  VerticalAlignment="Top" Margin="20,0">
<TextBlock Text="{x:Bind HeaderText}" Style="{ThemeResource SubheaderTextBlockStyle}"
  Margin="0,0,0,10" />
</StackPanel>
```

Then I use the overload for TryStart to reference both the connected animation and the UI element to coordinate with, like so:

```
ConnectedAnimation imageAnimation =
  ConnectedAnimationService.GetForCurrentView().GetAnimation("ca1");

if (imageAnimation != null)
{

imageAnimation.TryStart(detailedIamge, new UIElement[] { CoordinatedPanel });
}
```

This will allow both the connected animation I created and any other animations on the UI to run at the same time, helping the user understand context faster with a more immersive experience.

A quick note: I avoid using connected animations if my UI depends on network requests or any long-running asynchronous operations between preparing and starting the animation. These scenarios will cause a perceived glitch in your application or create delays that detract from the impact of the animation. To compensate for these situations, consider loading assets and images into your application ahead of time.

Wrapping Up

Swipe gesture actions and connected animations are useful resources that can ease interaction, add visual context, and create a compelling and intuitive experience for your end users. In the case of swipe gesture actions, the interaction pattern is easy to implement and adds a new level of efficiency for the user that extends to track pads and pen users. These little interactions can add up when users repeat the same actions over and over again.

Connected animations aid the user by providing visual context when navigating between pages, while at the same time enabling an engaging experience. From a developer standpoint, connected animations can be employed incrementally at key moments in the application. The result: End users enjoy a more cohesive and compelling experience that motivates them to use the application more often. ■

Lucas Haines *works with the XAML Controls team at Microsoft focusing on design and UI solutions for the Fluent Design System. He also worked three years at the Central Windows Design studio, where he helped shape the Fluent platform.*

Thanks to the following Microsoft technical experts for reviewing this article: *Steven Moyes, Kiki Saintonge*

Programming for watchOS with Xamarin and Visual Studio for Mac

Dawid Borycki

Small wearable devices like personal activity trackers and smart watches (such as Microsoft Band, Android Wear or Apple Watch) are becoming more and more popular. These wearables are equipped with various sensors, which monitor wearer's health parameters in real time. Many wearables also have communication interfaces, so sensor data can be easily transmitted to custom or dedicated cloud services (such as Microsoft Health) for storage or advanced processing. As a result, the wearable can act as an additional endpoint in an Internet of Things (IoT) ecosystem. This, in turn, can help advance personal health care to a new level, where IoT predictive algorithms can inform the user about emerging health issues in advance.

Wearables can also run custom apps. Developers are provided with dedicated SDKs. However, as is the case for many mobile devices, each platform has its own specific API, which can be accessed through platform-specific programming languages and tools. To make things easier, Xamarin provides support for Android Wear and watchOS within the Xamarin.Android and Xamarin.iOS libraries, respectively. You can develop wearable apps in a similar manner as mobile apps, by utilizing the common .NET code base, which is referenced in the platform-specific projects.

In this article, I'll show you how to utilize such an approach to build the watchOS app depicted in **Figure 1**. When you run this app, it begins retrieving the collection of objects from the REST Web service. This collection is composed of fake photos, each of which has a title and a bitmap (single color image). At this stage, the app only shows one button with the caption Get list. This button is disabled until the data is downloaded, as shown in the first row of **Figure 1**.

Tap the button and an action sheet (second row of **Figure 1**) appears displaying an alert that's composed of several buttons, defined as actions or action buttons (bit.ly/2EEUZpL). In this example, the action sheet provides the action buttons, whose captions contain the range of photos to be displayed. When you tap an action, the selected photos are displayed in the table control (bit.ly/2CaqOnM) just below the Get list button as shown in the last row in **Figure 1**. This table is scrollable, so you can scroll down the list to see all photos in the group.

I'll implement the communication with the Web service in a separate .NET Standard Class Library (bit.ly/2HArfMq). Following the referenced documentation, .NET Standard is a formal specification of the .NET APIs designed to accomplish uniform access to programming interfaces available on all .NET implementations. One of the main advantages of this approach is that a shared

This article discusses:

- Develop apps for Apple Watch with Xamarin and Visual Studio for Mac
- Learn how to reuse your existing C# code for wearables through .NET Standard
- Discover .NET Standard Class Libraries for code sharing

Technologies discussed:

Xamarin.iOS, Visual Studio for Mac, .NET Standard

library can be easily referenced in .NET projects to reduce or eliminate conditional compilation of the shared code. As a result, the .NET Standard Class Library can be implemented once and then referenced in various .NET projects, targeting Universal Windows Platform (UWP), .NET Core, ASP.NET Core, Xamarin.iOS, Xamarin.Android, Xamarin.Forms and so on without the need to recompile for each platform.

Here, I'll show how to reuse common code in the watchOS app. For the Web service I'll use the fake REST API server JSONPlaceholder (jsonplaceholder.typicode.com), which provides several resources, including the photos resource employed in the sample app. This resource stores a collection of fake pictures, each of which is represented as the following JSON object:

```
{
  "albumId": 1,
  "id": 1,
  "title": "accusamus beatae ad facilis cum similique qui sunt",
  "url": "http://placehold.it/600/92c952",
  "thumbnailUrl": "http://placehold.it/150/92c952"
}
```

Each photo has an associated id, photo album, title and two URLs pointing to a bitmap and its thumbnail. For this exercise, the bitmap is just a one-color image with a label showing the bitmap dimensions.

Everything you see here is created using Visual Studio for Mac. You can find the complete sample code for this project on GitHub at github.com/dawidborycki/Photos.

Parent App and Shared Code

The structure of a typical watchOS solution comprises three projects (see apple.co/2GwXhrn and bit.ly/2El2dNO). The first is the parent iOS app. Two other projects are dedicated to the watch app: Watch app bundle and WatchKit extension bundle. The parent iOS app is used as a proxy to deliver watch bundles to the wearable. The Watch app bundle contains interface storyboards. As is the case with iOS, developers use interface storyboards to define scenes and segues (transitions) between them. Finally, the WatchKit extension bundle contains resources and the app code.

Let's start by creating the parent iOS app using the new Single View iOS project, which can be found in the New Project creator of Visual Studio for Mac. I set the project and solution names to Photos.iOS and Photos, respectively. After creating the project, I supplement the Photos solution by adding another project Photos.Common, which I create using the .NET Standard Library project template. This template is located under the Multiplatform | Library group of the New Project creator.

When you create the .NET Standard Library you're given an option to choose the version of .NET Standard. This version determines the available API (the higher the version, the more functionality you can access) and supported platforms (the higher the version, the fewer supported platforms). Here, I set .NET Standard version to 2.0, which already includes the HttpClient class to communicate with the Web service over HTTP. You can retrieve the list of APIs for each .NET Standard version using the .NET API browser at bit.ly/2Fl44Fa.

After setting up the common project, I install one NuGet package—Newtonsoft.JSON—which will be used to deserialize HTTP responses.

Figure 1 **A Preview of the watchOS App**

To install the NuGet package in Visual Studio for Mac, you proceed as you would in Visual Studio for Windows. In the Solution Explorer, right-click Dependencies | NuGet node and from the context menu choose Add Packages. Visual Studio displays a window, in which you search for packages. Alternatively, you can use Package Console and install the NuGet package from the command line.

REST Client

Now I'm ready to implement the client class for the Photos Web service. To simplify deserialization, I map the JSON object, shown previously, to a C# class. This can be done manually or with a dedicated tool. Here, I'll use JSONUtils (jsonutils.com). This Web site has an intuitive interface that consists of the following elements:

- Class Name textbox where you enter your class name
- JSON Text or URL textbox, in which you place your JSON code or its URL

- Several radio buttons, which let you choose your language (C#, VB.NET and so on)
- Two checkboxes: Add Namespace and Pascal Case

To generate the C# class, I set the Class Name to Photo and paste the following URL for the JSON Text or URL textbox: jsonplaceholder.typicode.com/photos/1. Finally, I enable the Pascal Case checkbox and click the Submit button. The generated Photo class now appears at the bottom of the page. Photo class (see companion code: Photos.Common/Model/Photo.cs) does not require additional comments—it's just composed of the auto-implemented properties, representing the JSON object shown previously.

To implement the REST client, I create the static PhotoClient class (Photos.Common project). This class has one field of type HttpClient. This field is instantiated within the static constructor to set the BaseAddress property such that it points to the JSONPlaceholder URL, as shown here:

```
private static HttpClient httpClient;

static PhotosClient()
{
  httpClient = new HttpClient()
  {
    BaseAddress = new Uri("https://jsonplaceholder.typicode.com/")
  };
}
```

As **Figure 2** shows, the PhotosClient has two public methods:

- GetByAlbumId: Achieves the collection of photos from the given album using the GetAsync method of the HttpClient class. After checking the HTTP response status code (CheckStatusCode), the resulting response is deserialized to a collection of C# Photo objects using a generic helper method DeserializeResponse (helper methods are discussed later).
- GetImageData: Retrieves the byte array, representing the photo from the provided URL. To get the image data I use GetByteArrayAsync of the HttpClient class instance.

PhotosClient also implements two private methods: CheckStatusCode and DeserializeResponse. The first method accepts an instance of the HttpResponseMessage class and checks the value of its IsSuccessStatusCode property, throwing an exception if the status code was different than 200 (success code).

```
private static void CheckStatusCode(HttpResponseMessage response)
{
  if (!response.IsSuccessStatusCode)
  {
    throw new Exception($"Unexpected status code: {response.StatusCode}");
  }
}
```

The second method, DeserializeReponse, also accepts an instance of the HttpResponseMessage class, but reads the message body as a string using HttpResponseMessage.Content.ReadAsStringAsync method. The resulting value is then passed to the static DeserializeObject method of the Newtonsoft.Json.JsonConvert class. The latter returns the C# object of the given type, as shown here:

```
private static async Task<T> DeserializeResponse<T>(HttpResponseMessage response)
{
  var jsonString = await response.Content.ReadAsStringAsync();

  return JsonConvert.DeserializeObject<T>(jsonString);
}
```

In **Figure 2** I also use the IsNull method of the custom Check class. IsNull method performs simple argument validation to check whether argument is null. If so, an exception of type ArgumentNullException will be thrown (see companion code: Photos.Common/Helpers/Check.cs).

The shared functionality is now ready, so I can proceed to implement the watchOS app.

The watchOS App and Its Structure

To create the actual watchOS app I first right-click the solution name (in this case Photos) under the Solution Explorer and then choose Add | Add New Project from the context menu. A New Project dialog appears, in which I click the following tab: watchOS | App (make sure you don't use an Extension or Library tab). A list of project templates appears, from which I choose WatchKit App C# project template. Afterward, the list of configurable options will be presented, as shown in **Figure 3**.

As mentioned previously, every watchOS app has an associated parent iOS app. In this instance it's the Photos.iOS app. Then, in the App Name textbox I type WatchKit, set the Target to watchOS 4.2 (note that the particular items in this list will depend on the installed SDK version) and uncheck all checkboxes under the Scenes group.

After clicking the Next button, Visual Studio will create two additional projects: Photos.iOS.WatchKit and Photos.iOS.WatchKitExtension.

The first project (WatchKit) contains the storyboard, which you use to define scenes and segues (transitions) between them. The second project (WatchKitExtension) contains associated logic, including view controllers that in the watchOS are called interface controllers. So, you typically modify the UI of the watch app using storyboard designer (shown in **Figure 4**), which is activated by double-clicking the Interface.storyboard file of the WatchKit app.

Figure 2 **Public Methods of the PhotoClient Class**

```
public static async Task<IEnumerable<Photo>> GetByAlbumId(int albumId)
{
  var response = await httpClient.GetAsync($"photos?albumId={albumId}");

  var photoCollection = new List<Photo>();

  try
  {
    CheckStatusCode(response);

    photoCollection = await DeserializeResponse<List<Photo>>(response);
  }
  catch(Exception ex)
  {
    Console.WriteLine(ex.Message);
  }

  return photoCollection;
}

public static async Task<byte[]> GetImageData(Photo photo)
{
  var imageData = new byte[0];

  try
  {
    Check.IsNull(photo);

    imageData = await httpClient.GetByteArrayAsync(photo.ThumbnailUrl);
  }
  catch(Exception ex)
  {
    Console.WriteLine(ex.Message);
  }

  return imageData;
}
```

DocuVieware 3

HTML5 Viewer & Document Management Kit

NEW RELEASE

Easy integration

Full support for custom snap-in

Zero-footprint solution

Fully customizable UI

Mobile devices optimization

Fast & crystal-clear rendering

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

DOWNLOAD
YOUR FREE TRIAL

www.docuvieware.com

Powered by **GdPicture.NET** 14

DocuVieware is an ORPALIS imaging technologies product

The storyboard designer lets you drag and drop controls from the Toolbox onto the scenes. In **Figure 4** I have just one scene, which is associated with the InterfaceController class, defined under the Photos.iOS.WatchKit-Extension project. This class is like the UIViewController for an iOS app—that is, it presents and manages content on the screen and implements methods that handle user interactions. Note that once controls are added to the scene, you can modify their properties using the Properties pad and then access them from the code through the InterfaceController class.

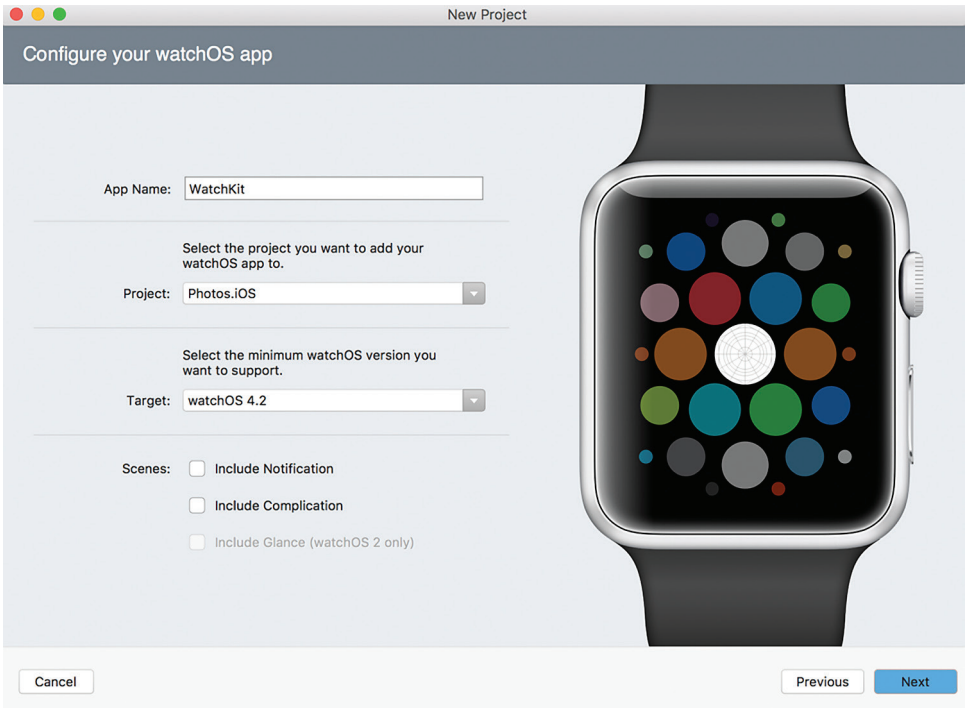


Figure 3 **Configuring the watchOS App**

Before editing the UI, let's briefly investigate the structure of this class, which derives from the WatchKit.WKInterfaceController—the base class for all interface controllers. A default implementation of the InterfaceController overrides three methods related to the view lifecycle (see apple.co/2GwXhrn):

- **Awake:** Invoked by the system right after the InterfaceController is initialized. You typically use this method to load data and prepare the UI.
- **WillActivate:** Invoked when the associated view is about to become active. You use this method to prepare final updates right before the interface will be displayed.
- **DidDeactivate:** Called when the view becomes inactive. You typically use this method to release dynamic resources, which aren't needed anymore.

These methods are used to configure your view, depending on its visibility. To give you a simple example, I'll analyze the following Awake method:

```
public override void Awake(
  NSObject context)
{
  base.Awake(context);

  SetTitle("Hello, watch!");
}
```

This code first invokes the Awake method of the base class (WKInterfaceController) and then calls the SetTitle method, which changes the string displayed in the top-left corner of the view. This title is a default element of each interface controller.

To test this modification, you can execute the Photos.iOS.WatchKit app in the simulator. First, you need to set this app (not the extension bundle) as the startup project, using a dropdown from the toolbar in Visual Studio for Mac. Next to that list you have two other dropdown lists—one for selecting the configuration (Debug or Release), and the other for selecting from a list of simulators. Here, I'm using the Debug configuration and the Apple Watch Series 3 – 42 mm – watchOS 4.2 emulator. After choosing the simulator

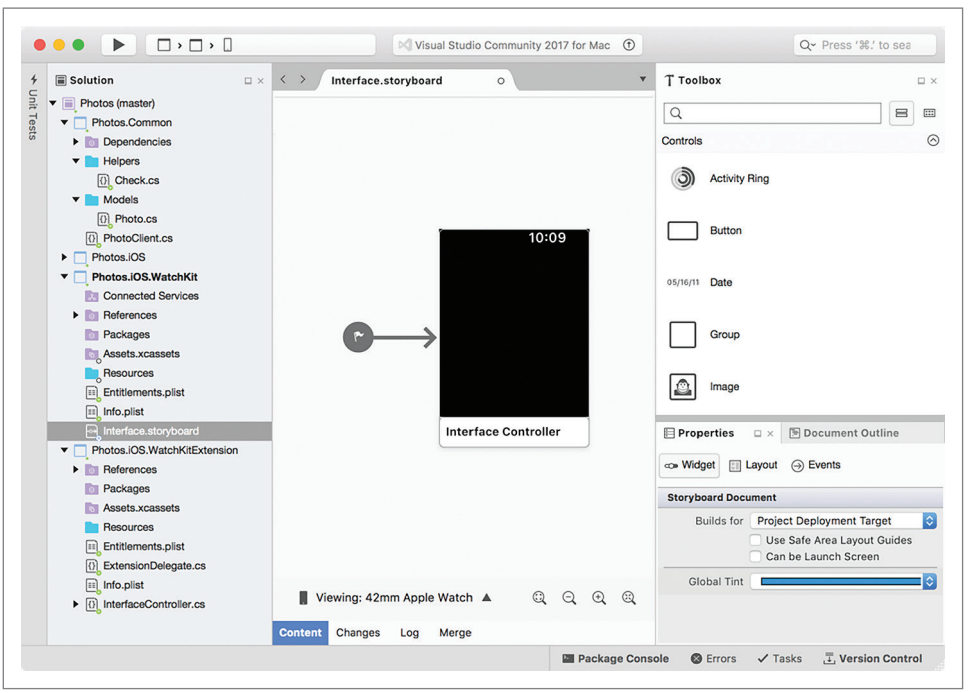


Figure 4 **Designing the UI of the watchOS App**

Manipulating Documents?

APIs to view, convert, annotate, compare, sign, assemble and search documents in your applications.

Try GroupDocs APIs for FREE

GROUPDOCS
Document Manipulation APIs

Microsoft .NET

GroupDocs.Viewer

View over 50 documents and image formats in any application using document viewer APIs.

GroupDocs.Annotation

Add annotations to specific words, phrases and any region of the document.

GroupDocs.Conversion

Fast batch document conversion APIs for any .NET, Java or Cloud app.

GroupDocs.Comparison

Compare two documents and get a difference summary report.

GroupDocs.Signature

Digitally sign Microsoft Word, Excel, PowerPoint and PDF documents.

GroupDocs.Assembly

Document automation APIs to create reports from templates and various data sources.

- GroupDocs.Metadata
- GroupDocs.Search
- GroupDocs.Text
- GroupDocs.Editor

Download a Free Trial at
https://downloads.groupdocs.com

Americas: +1 903 306 1676
EMEA: +44 141 628 8900
Oceania: +61 2 8006 6987
sales@asposeptyltd.com

Figure 5 **Partitioning Photo Titles**

```
private const int rowsPerGroup = 10;
private IEnumerable<Photo> photos;
private WKAlertAction[] alertActions;

private void CreateAlertActions()
{
  var actionsCount = photos.Count() / rowsPerGroup;

  alertActions = new WKAlertAction[actionsCount];

  for (var i = 0; i < actionsCount; i++)
  {
    var rowSelection = new RowSelection(
      i, rowsPerGroup, photos.Count());

    var alertAction = WKAlertAction.Create(
      rowSelection.Title,
      WKAlertActionStyle.Default,
      async () => { await DisplaySelectedPhotos(rowSelection); });

    alertActions[i] = alertAction;
  }
}
```

and clicking the Play icon, the app compiles and deploys. Note that two simulators are launched: iOS simulator and its paired watchOS simulator (refer back to the left screen in **Figure 1**).

Action Sheet

Now I can move forward to implement the actual UI of the Photos.iOS.WatchKit app. As shown in **Figure 1**, the UI of the app comprises three elements: a button, an action sheet and a table view. When the user taps the button, an action sheet is activated. It presents several options, which let the user choose the group of photos to be displayed in the table view. I implemented this photo grouping to comply with Apple's guidance to limit the number of rows in the table view as a way to improve app performance (apple.co/2Cecrnt).

I'll start by creating the list of buttons, which will be displayed in the action sheet. The list of these buttons is generated based on the photo collection (photos field) retrieved from the Web service as shown in **Figure 5**. Each action button is represented as an instance of the WatchKit.WKAction class. This class lacks any public constructors but implements the static Create method, which you use to create actions. As shown in **Figure 5**, the Create method accepts three arguments:

- Title defines the action button caption
- Style indicates the style of action button
- Handler specifies a method to be executed when the user taps the action button

In **Figure 5** all actions have a default style, represented as the Default value from the WatchKit.WKAlertStyle enumeration. This enumeration defines two other values (apple.co/2EHCAZr): Cancel and Destructive. You use the first one to create an action that cancels an operation without any changes. The destructive style should be applied to actions that produce irreversible modifications.

The CreateAlertActions method from **Figure 5** partitions the photos into chunks, each of which contains 10 elements (rowsPerGroup constant). To get a selected group of photos from the collection, I need two indices—one where the group begins (beginIndex variable) and one where it ends (endIndex). To calculate these indices, I use the RowSelection class, which is also used to create titles for items displayed in the action sheet. RowSelection class is implemented in the Photos.iOS.WatchKitExtension project (RowSelection.cs) and its most important parts are shown in **Figure 6**.

Finally, each button of the action sheet has an associated handler, which invokes the DisplaySelectedPhotos method. This method is responsible for presenting the table of selected photos and will be described later.

To activate the action sheet I first reference the Photos.Common project. To do so, in Solution Explorer I right-click References of the Photos.iOS.WatchKitExtension, select Edit References, choose the Project tab and select Photos.Common. Once in the Reference Manager I also need to reference Newtonsoft.Json.dll library to ensure that it will be copied to the output directory. I do this using the .NET Assembly tab, clicking the Browse button and then choosing Newtonsoft.Json.dll from the folder packages/Newtonsoft.Json/lib/netstandard20. This folder is created after installing the Newtonsoft.Json NuGet package.

These steps are required to access a shared code base (including PhotosClient that was implemented earlier) from the watchOS app. I then modify the UI using the storyboard. A detailed description of how layout works in watchOS can be found in Apple (apple.co/2FIzADj) and Xamarin documentation (bit.ly/2EKjCRM).

After opening the storyboard designer, I drag the Button control from the Toolbox onto the scene. Using the Properties pad, I set the button's Name and Title properties to ButtonDisplayPhotoList and Get list, respectively. Then, I create the event handler, which executes whenever the user taps the button. To create an event handler, I use the Properties pad, clicking the Events tab and then typing ButtonDisplayPhotoList_Activated in the Action search box. After pressing the enter key, Visual Studio declares the new method in the InterfaceController class. Finally, the ButtonDisplayPhotoList_Activated is defined as follows:

```
partial void ButtonDisplayPhotoList_Activated()
{
  PresentAlertController(string.Empty,
    string.Empty,
    WKAlertControllerStyle.ActionSheet,
    alertActions);
}
```

To create and present an action sheet, I use PresentAlertController. This method accepts four arguments:

- **Title** indicates the title of the alert.
- **Message** specifies the text to be displayed in the alert's body.
- **PreferredStyle** specifies the style of the alert controller. Style

Figure 6 **Calculating Indices with RowSelection Class**

```
public int BeginIndex { get; private set; }
public int EndIndex { get; private set; }
public int RowCount { get; private set; }
public string Title { get; private set; }

private static string titlePrefix = "Elements to show:";

public RowSelection(int groupIndex, int rowsPerGroup, int elementCount)
{
  BeginIndex = groupIndex * rowsPerGroup;
  EndIndex = Math.Min((groupIndex + 1) * rowsPerGroup, elementCount) - 1;

  RowCount = EndIndex - BeginIndex + 1;
  Title = $"{titlePrefix} {BeginIndex}-{EndIndex}";
}
```

is represented by one of the values, defined in the WatchKit.WKAlertControllerStyle enumeration: Alert, SideBySideButtonsAlert or ActionSheet. Differences between them are summarized at apple.co/2GA57Rp.

- **Actions** are a collection of action buttons to be included in the alert. Note that the number of actions depends on the alert style as described in the referenced documentation.

Here, both title and message are set to string.Empty, while the alert style is set to ActionSheet. As a result, only action buttons will be displayed (refer back to **Figure 1**). To ensure that alertActions are ready before the user taps the Get list button, I retrieve photos and titles within the Awake method (as shown in **Figure 7**):

You can now run the app. When the photo album is retrieved from the remote server, the button will be enabled. Click it to activate the action sheet (shown in the middle part of **Figure 1**).

Table View

To complete this implementation, I need to create the table view showing the selected group of photos. To that end I open the storyboard designer and from the Toolbox drag the Table control onto the scene (I'll place it just below the Get list button).

In the next step, I need to define the cell layout. By default, this layout has a single control: Group. I can use this as the parent for other controls, but first I need to ensure that the group control is active. So, I click the Document Outline pad (visible in the bottom part of **Figure 4**) and then click Group Control under the Table/Table Row. Next, I drag Image and Label controls onto the table. Image and Label will appear in the table view and also in the Document Outline. I'll configure all control properties as follows: **Image** (Name: ImagePhotoPreview; Size: Fixed Width and Height of 50 px), **Label** (Name: LabelPhotoTitle), **Table** (Name: TablePhotos), **Group** (Size: Fixed height of 50 px).

It's important to explicitly set control names so you can easily refer to them from the code. Once you specify the control name, Visual Studio makes an appropriate declaration under the InterfaceController.designer.cs file.

In watchOS every row has a dedicated row controller, which you use to control row appearance. Here, I'll use this controller to specify content for the image and label of each row. To create the row controller, select the Table Row in the Document Outline pad, and then open the Properties tab, where you type PhotoRowController in the Class textbox. A new file, PhotoRowController.cs, will be added. It contains the class of the same name. I then supplement the definition of this class by another method, as follows:

```
public async Task SetElement(Photo photo)
{
  Check.IsNull(photo);

  // Retrieve image data and use it to create UIImage
  var imageData = await PhotosClient.GetImageData(photo);
  var image = UIImage.LoadFromData(NSData.FromArray(imageData));

  // Set image and title
  ImagePhotoPreview.SetImage(image);
  LabelPhotoTitle.SetText(photo.Title);
}
```

The SetElement function accepts one argument of type Photo to display a photo thumbnail along with photo title in the appropriate row of table view. Then, to actually load and configure table rows, I extend the definition of the InterfaceController using the following method:

```
private async Task DisplaySelectedPhotos(RowSelection rowSelection)
{
  TablePhotos.SetNumberOfRows(rowSelection.RowCount, "default");

  for (int i = rowSelection.BeginIndex, j = 0;
    i <= rowSelection.EndIndex; i++, j++)
  {
    var elementRow = (PhotoRowController)TablePhotos.GetRowController(j);

    await elementRow.SetElement(photos.ElementAt(i));
  }
}
```

RowSelection is passed to DisplaySelectedPhotos method in order to provide necessary information about the rows to be displayed. More specifically, RowCount property is used to set the number of rows to be added to the table (TablePhotos.SetNumberOfRows). Subsequently, DisplaySelectedPhotos iterates through table rows to set the content for each row. At each iteration I first obtain a reference to the PhotoRowController associated with the current row. Given that reference I invoke the PhotoRowController.SetElement method in order to get image data and title, which are displayed in the table cell.

Finally, after running the app, you'll get the results shown previously in **Figure 1**.

Figure 7 **Retrieve Photos and Titles**

```
public async override void Awake(NSObject context)
{
  base.Awake(context);

  SetTitle("Hello, watch!");

  // Disable button until the photos are downloaded
  ButtonDisplayPhotoList.SetEnabled(false);

  // Get photos from the web service (first album only)
  photos = await PhotosClient.GetByAlbumId(1);

  // Create actions for the alert
  CreateAlertActions();

  ButtonDisplayPhotoList.SetEnabled(true);
}
```

Wrapping up

In this article, I showed how to develop watchOS apps with Xamarin, Visual Studio for Mac and a shared C# .NET code base implemented within the .NET Standard Class Library. Along the way I explored some of the most important elements of watchOS apps, including the app structure, interface controllers and selected UI controls (button, action sheet, table view). Because the shared code base is implemented using the same approach employed with mobile apps, you can easily extend your mobile solution to target smart wearables. You can get more info on watchOS from the Apple at apple.co/2EFLeaL, as well as detailed Xamarin documentation at bit.ly/2ohSwLU. ■

Dawid Borycki *is a software engineer and biomedical researcher, author and conference speaker. He enjoys learning new technologies for software experimenting and prototyping.*

Thanks to the following technical expert for reviewing this article: *Brad Umbaugh*

AUGUST 6 – 10, 2018 • Microsoft Headquarters, Redmond, WA

Change is ~~Coming.~~ Are You Ready? HERE

Join us for TechMentor, August 6 – 8, 2018, as we return to Microsoft Headquarters in Redmond, WA. In today's IT world, more things change than stay the same. As we celebrate the 20th year of TechMentor, we are more committed than ever to providing immediately usable IT education, with the tools you need today, while preparing you for tomorrow – **keep up, stay ahead and avoid Winter, ahem, Change.**

Plus you'll be at the source, Microsoft HQ, where you can have lunch with Blue Badges, visit the company store, and experience life on campus for a week!

You owe it to yourself, your company and your career to be at TechMentor Redmond 2018!

SAVE $400!
REGISTER NOW
Use Promo Code TMAPR

EVENT PARTNER
Microsoft

SUPPORTED BY
Redmond
Redmond Channel Partner
Virtualization & Cloud Review

PRODUCED BY
1105 MEDIA
YOUR GROWTH, OUR BUSINESS.

AGENDA AT-A-GLANCE

Client and Endpoint Management	PowerShell and DevOps	Infrastructure	Soft Skills for ITPros	Security	Cloud (Public/ Hybrid/Private)

START TIME	END TIME	TechMentor Pre-Conference Workshops: Monday, August 6, 2018 *(Separate entry fee required)*			
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Light Breakfast			
9:00 AM	12:00 PM	**M01** Workshop: How to Prevent all Ransomware / Malware in 2018 - *Sami Laiho*	**M02** Workshop: Building Office 365 Federated Identity from Scratch Using AD FS - *Nestori Syynimaa*	**M03** Workshop: Managing Windows Server with Project Honolulu - *Dave Kawula*	
12:00 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
2:00 PM	5:00 PM	**M01** Workshop: How to Prevent all Ransomware / Malware in 2018 (Continues) - *Sami Laiho*	**M04** Workshop: Master PowerShell Tricks for Windows Server 2016 and Windows 10 - *Will Anderson & Thomas Rayner*	**M05** Workshop: Dave Kawula's Notes from the Field on Microsoft Storage Spaces Direct - *Dave Kawula*	
6:30 PM	8:30 PM	Dine-A-Round Dinner - Suite in Hyatt Regency Lobby			
START TIME	**END TIME**	**TechMentor Day 1: Tuesday, August 7, 2018**			
7:00 AM	8:00 AM	Registration - Coffee and Light Breakfast			
8:00 AM	9:15 AM	**T01** Enterprise Client Management in a Modern World - *Kent Agerlund*	**T02** How to Write (PowerShell) Code that Doesn't Suck - *Thomas Rayner*	**T03** The Easy Peasy of Troubleshooting Azure - *Mike Nelson*	**T04** Nine O365 Security Issues Microsoft Probably Hasn't Told You (and You Probably Don't Want to Know) - *Nestori Syynimaa*
9:30 AM	10:45 AM	**T05** Managing Client Health—Getting Close to the Famous 100% - *Kent Agerlund*	**T06** The Network is Slow! Or is it? Network Troubleshooting for Windows Administrators - *Richard Hicks*	**T07** Getting Started with PowerShell 6.0 for IT Pro's - *Sven van Rijen*	**T08** The Weakest Link of Office 365 Security - *Nestori Syynimaa*
11:00 AM	12:00 PM	**KEYNOTE: To Be Announced** - *Stephen L. Rose, Sr. PMM, One Drive For Business, Microsoft*			
12:00 PM	1:00 PM	Lunch - *McKinley* / Visit Exhibitors - *Foyer*			
1:00 PM	2:15 PM	**T09** How to Get Started with Microsoft EMS Right Now - *Peter Daalmans*	**T10** Back to the Future! Access Anywhere with Windows 10 Always on VPN - *Richard Hicks*	**T11** Using Desired State Configuration in Azure - *Will Anderson*	**T11** To Be Announced
2:15 PM	2:45 PM	Sponsored Break - Visit Exhibitors - *Foyer*			
2:45 PM	4:00 PM	**T12** Conceptualizing Azure Resource Manager Templates - *Will Anderson*	**T13** How to Use PowerShell to Become a Windows Management SuperHero - *Petri Paavola*	**T14** Making the Most Out of the Azure Dev/Test Labs - *Mike Nelson*	**T15** To Be Announced
4:00 PM	5:30 PM	Exhibitor Reception – Attend Exhibitor Demo - *Foyer*			
START TIME	**END TIME**	**TechMentor Day 2: Wednesday, August 8, 2018**			
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast			
8:00 AM	9:15 AM	**W01** Automated Troubleshooting Techniques in Enterprise Domains (Part 1) - *Petri Paavola*	**W02** Troubleshooting Sysinternals Tools 2018 Edition - *Sami Laiho*	**W03** In-Depth Introduction to Docker - *Neil Peterson*	**W04** How Microsoft Cloud Can Support Your GDPR Journey - *Milad Aslaner*
9:30 AM	10:45 AM	**W05** Automated Troubleshooting Techniques in Enterprise Domains (Part 2) - *Petri Paavola*	**W06** What's New in Windows Server 1803 - *Dave Kawula*	**W07** Simplify and Streamline Office 365 Deployments the Easy Way - *John O'Neill, Sr.*	**W08** How to Administer Microsoft Teams Like a Boss - *Ståle Hansen*
11:00 AM	12:00 PM	**TECHMENTOR PANEL: The Future of Windows** - *Peter De Tender, Dave Kawula, Sami Laiho, & Petri Paavola*			
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch - *McKinley* / Visit Exhibitors - *Foyer*			
1:00 PM	1:30 PM	Networking Break - Exhibitor Raffle @ 1:10 pm (Must be present to win) - *Foyer in front of Business Center*			
1:30 PM	2:45 PM	**W09** Putting the Windows Assessment and Deployment Kit to Work - *John O'Neill, Sr.*	**W10** Deploying Application Whitelisting on Windows Pro or Enterprise - *Sami Laiho*	**W11** Azure is 100% High-Available... Or Is It? - *Peter De Tender*	**W12** What the NinjaCat Learned from Fighting Cybercrime - *Milad Aslaner*
3:00 PM	4:15 PM	**W13** The Evolution of a Geek—Becoming an IT Architect - *Mike Nelson*	**W14** Advanced DNS, DHCP and IPAM Administration on Windows Server 2016 - *Orin Thomas*	**W15** Managing Tesla Vehicles from the Cloud - *Marcel Zehner*	**W16** Nano Server—Containers in the Cloud - *David O'Brien*
6:15 PM	9:00 PM	Set Sail! TechMentor's Seattle Sunset Cruise - *Busses depart the Hyatt Regency at 6:15pm to travel to Kirkland City Dock*			
START TIME	**END TIME**	**TechMentor Day 3: Thursday, August 9, 2018**			
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast			
8:00 AM	9:15 AM	**TH01** Manage Your Apple Investments with Microsoft EMS - *Peter Daalmans*	**TH02** Tips and Tricks for Managing and Running Ubuntu/Bash/Windows Subsystem for Linux - *Orin Thomas*	**TH03** The OMS Solutions Bakery - *Marcel Zehner*	**TH04** Getting Started with PowerShell for Office 365 - *Vlad Catrinescu*
9:30 AM	10:45 AM	**TH05** HoloLens, Augmented Reality, and IT - *John O'Neill, Sr.*	**TH06** A Real-world Social Engineering Attack and Response - *Milad Aslaner*	**TH07** 30 Terrible Habits of Server and Cloud Administrators - *Orin Thomas*	**TH08** Advanced PowerShell for Office 365 - *Vlad Catrinescu*
11:00 AM	12:15 PM	**TH09** 10 Tips to Control Access to Corporate Resources with Enterprise Mobility + Security - *Peter Daalmans*	**TH10** What's New and Trending with Microsoft Enterprise Client Management - *Kent Agerlund*	**TH11** OneNote LifeHack: 5 Steps for Succeeding with Personal Productivity - *Ståle Hansen*	**TH12** Managing Virtual Machines on AWS—Like in Real Life! - *David O'Brien*
12:15 PM	2:15 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
2:15 PM	3:30 PM	**TH13** Security Implications of Virtualizing Active Directory Domain Controllers - *Sander Berkouwer*	**TH14** Building a New Career in 5 Hours a Week - *Michael Bender*	**TH15** Azure CLI 2.0 Deep Dive - *Neil Peterson*	**TH16** OpenSSH for Windows Pros - *Anthony Nocentino*
3:45 PM	5:00 PM	**TH17** Running Hyper-V in Production for 10 years - Notes from the Field - *Dave Kawula*	**TH18** Network Sustainability and Cyber Security Measures - *Omar Valerio*	**TH19** Azure AD Connect Inside and Out - *Sander Berkouwer*	**TH20** I Needed to Install 80 SQL Servers...Fast. Here's How I Did It! - *Anthony Nocentino*
START TIME	**END TIME**	**TechMentor Post-Conference Workshops: Friday, August 10, 2018** *(Separate entry fee required)*			
8:30 AM	9:00 AM	Post-Conference Workshop Registration - Coffee and Light Breakfast			
9:00 AM	12:00 PM	**F01** Workshop: Hardening Your Windows Server Environment - *Orin Thomas*		**F02** Workshop: Learn the Latest and Greatest Updates to the Azure Platform IaaS and PaaS Services v2.0 - *Peter De Tender*	
12:00 PM	1:00 PM	Lunch - *McKinley*			
1:00 PM	4:00 PM	**F01** Workshop: Hardening Your Windows Server Environment (Continues) - *Orin Thomas*		**F02** Workshop: Learn the Latest and Greatest Updates to the Azure Platform IaaS and PaaS Services v2.0 (Continues) - *Peter De Tender*	

Speakers and sessions subject to change

DINO ESPOSITO

Discovering ASP.NET Core SignalR

ASP.NET SignalR was introduced a few years ago as a tool for ASP.NET developers to add real-time functionality to applications. Any scenarios in which an ASP.NET-based application had to receive frequent and asynchronous updates from the server—from monitoring systems to gaming—were good use cases for the library. Over the years, I used it also to refresh the UI in CQRS architecture scenarios and to implement a Facebook-like notification system in socialware applications. From a more technical perspective, SignalR is an abstraction layer built over some of the transport mechanisms that can establish a real-time connection between a fully compatible client and server. The client is often a Web browser and the server is often a Web server, but both are not limited to that.

ASP.NET SignalR is part of ASP.NET Core 2.1. The overall programming model of the library is similar to that for classic ASP.NET, but the library itself has in fact been completely rewritten. Still, developers should quickly become comfortable with the new scheme, once they adjust to the changes that exist here and there. In this article, I'll discuss how to use the new library in a canonical Web application to monitor a remote, and possibly lengthy, task.

Setup of the Environment

You may need a couple of NuGet packages to use the library: Microsoft.AspNetCore.SignalR and Microsoft.AspNetCore.SignalR.Client. The former provides the core functionality; the latter package is the .NET client and is needed only if you're building a .NET Client application. In this case, we're using it through a Web client, so a SignalR NPM package would be needed instead. I'll discuss the details of this later in the article. Note that using SignalR in the context of a Web application based on the MVC application model is not a requirement. You can use the services of the SignalR library directly from an ASP.NET Core console application and can also host the server part of SignalR in a console application.

Figure 1 **The Startup Class of a SignalR ASP.NET Core Application**

```
public class Startup
{
  public void ConfigureServices(IServiceCollection services)
  {
    services.AddMvc();
    services.AddSignalR();
  }

  public void Configure(IApplicationBuilder app)
  {
    app.UseStaticFiles();
    app.UseDeveloperExceptionPage();

    // SignalR
    app.UseSignalR(routes =>
    {
      routes.MapHub<UpdaterHub>("/updaterDemo");
      routes.MapHub<ProgressHub>("/progressDemo");
    });

    app.UseMvcWithDefaultRoute();
  }
}
```

Not surprisingly, the startup class of the application needs to contain some specific code. In particular, you'll add the SignalR service to the collection of system services and configure it for actual use. **Figure 1** presents the typical status of a startup class that uses SignalR.

The configuration of the SignalR service consists in the definition of one or more server routes that bind to one or more endpoints within the server-side environment. The MapHub<T> method links the specified names, which will be part of a requesting URL, to an instance of a hub class. The hub class is the beating heart of the implementation of the SignalR protocol and is where the client calls are handled. You create a hub for each logically related group of calls the server-side intends to accept and handle. A SignalR conversation is made of messages being exchanged between two parties and each party may call methods on the other party receiving no response, one or multiple responses, or just the notification of an error. Any ASP.NET Core SignalR server implementation will expose one or more hubs. In **Figure 1**, you have two hub classes—UpdaterHub and ProgressHub—bound to unique strings that will be internally used to compose the URL target of the actual calls.

The Hub Class

The hub class in a SignalR application is a plain, simple class that inherits from the base hub class. The base class has the sole purpose of saving developers from writing over and over again the same boilerplate code. The base class only provides some infrastructure but no predefined behavior. In particular, it defines the members in **Figure 2**.

The simplest hub class is as minimal as the following:

```
public class ProgressHub : Hub
{
}
```

Interestingly enough, that's just the form of the hub if you use it from within a controller method in the context of an ASP.NET Core MVC application. Almost all of the examples you may find about ASP.NET Core SignalR (including the chat example) tend

Figure 2 **Members of the Hub Base Class**

Member	Description
Clients	Property that exposes the current list of clients managed by the hub.
Context	Property that exposes the current caller context, including information such as the ID of the connection and the claims of the user, if available.
Groups	Property that exposes the various subsets of clients that may have been programmatically defined as groups within the full list of clients. A group is typically created as a way to broadcast specific messages to a selected audience.
OnConnectedAsync	Virtual method invoked whenever a new client connects to the hub.
OnDisconnectedAsync	Virtual method invoked whenever a new client disconnects to the hub.

to show a direct and bidirectional binding between the client and the hub without any sort of intermediation by a controller. In this case, the hub will take a slightly more shaped form:

```
public class SampleChat : Hub
{
  // Invoked from outside the hub
  public void Say(string message)
  {
    // Invoke method on listening client(s)
    return Clients.All.InvokeAsync("Said", message);
  }
}
```

Unlike the canonical SignalR chat example that's rehashed in dozens of blog posts, the example I'll present here doesn't really set up a bidirectional conversation between client and server. The connection is established from the client but after that the client won't send any further requests. The server, instead, will be monitoring the progress of a task and pushing back data to the client whenever appropriate. In other words, the hub class needs to have public methods as the previous code only if the use case requires that the client directly calls into them. If it seems a bit obscure, the following example will shed enough light.

Monitoring a Remote Task

Here's the scenario: An ASP.NET Core application presents the user some HTML interface for the user to trigger a remote task (say, the creation of a report) that can be lengthy. Because of this, as a developer you want to display a progress bar to provide continuous feedback (see **Figure 3**).

As you can guess, in this example both the client and the server are setting up a SignalR conversation live in the context of the same ASP.NET Core project. At this stage of development, you have a fully functional MVC project just extended with the startup code of **Figure 1**. Let's set up the client framework. You need to have this setup work done in every Razor (or plain HTML) view that interacts with a SignalR endpoint.

To communicate with a SignalR endpoint from within a Web browser, the first thing you do is add a reference to the SignalR JavaScript client library:

```
<script src="~/scripts/signalr.min.js">
</script>
```

You can get this JavaScript file in a number of ways. The most recommended is via the Node.js Package Manager (NPM) tool that's available on nearly any development machine, especially after Visual Studio 2017. Through NPM, you look for the ASP.NET Core SignalR client named @aspnet/signalr and install it. It copies a bunch of JavaScript files to your disk, only one of which is strictly needed in most scenarios. Anyway, it's a simple matter of linking a JavaScript file and you can get that file in many other ways, including copying it from an older ASP.NET Core SignalR project. However, NPM is the only supported way the team provides for getting the script. Note also that ASP.NET Core SignalR no longer depends on jQuery.

In the client application, you also need another, more specific, segment of JavaScript code. In particular, you need code like this:

```
var progressConnection =
  new signalR.HubConnection("/progressDemo");
progressConnection.start();
```

You create a connection to the SignalR hub that matches the specified path. More precisely, the name you pass as an argument to HubConnection should be one of the names you mapped to a route in the startup class. Internally, the HubConnection object prepares a URL string that results from the concatenation of the current server URL and the given name. That URL is processed only if it matches one of the configured routes. Note also that if client and server are not the same Web application, then HubConnection must be passed the full URL of the ASP.NET Core application that hosts the SignalR hub, plus the hub name.

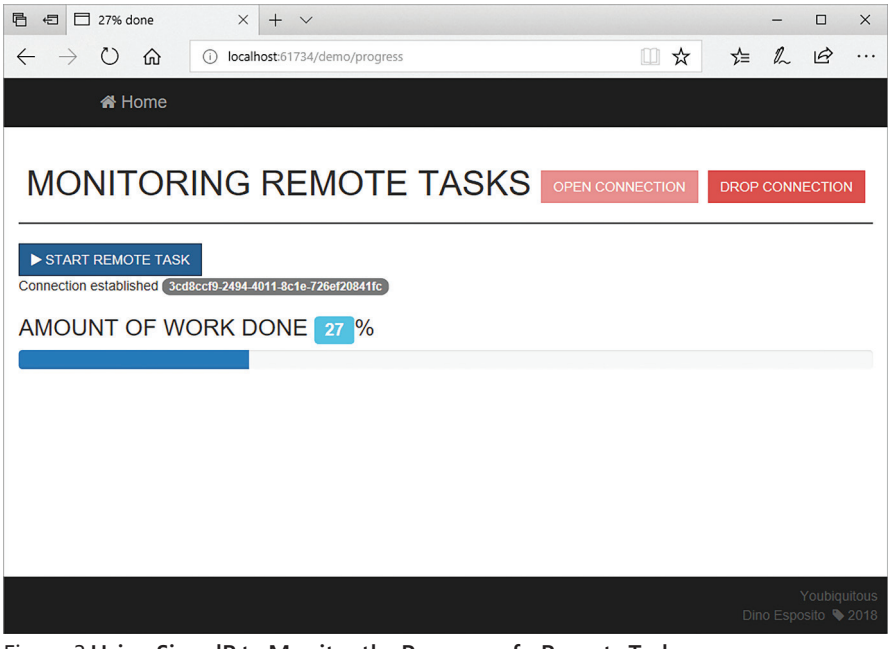


Figure 3 **Using SignalR to Monitor the Progress of a Remote Task**

The JavaScript hub connection object must then be opened up via the start method. You can use JavaScript promises—specifically the then method—or async/await in TypeScript to perform subsequent actions such as initializing some user interface. A SignalR connection is uniquely identified by a string ID.

It's important to notice that ASP.NET Core SignalR no longer supports automatic reconnect if the transport connection or the server fails. In older versions, in case of server failure the client attempts to re-establish a connection according to a scheduling algorithm and, if successful, it reopens a connection with the same ID. In SignalR Core if the connection drops the client can only start it again via the method start and this results in a different connection instance over a different connection ID.

The Client Callback API

Another fundamental segment of JavaScript code you need is the JavaScript that will be called back by the hub to refresh the interface and reflect on the client that progress is being made on the server. The way you write this code is a bit different in ASP.NET Core SignalR from what it was in older versions, but the intent is exactly the same. In our example, we have three methods that could be called back from the server: initProgressBar, updateProgressBar and clearProgressBar. Needless to say, names and signatures are arbitrary. Here's a sample implementation:

```
progressConnection.on("initProgressBar", () => {
  setProgress(0);
  $("#notification").show();
});
progressConnection.on("updateProgressBar", (perc) => {
  setProgress(perc);
});
progressConnection.on("clearProgressBar", () => {
  setProgress(100);
  $("#notification").hide();
});
```

For example, when the initProgressBar method is called back from the server the helper setProgress JavaScript function will configure the progress bar (the demo uses a Bootstrap progress bar component) and show it. Note that the jQuery library is used in the code but only to update the UI. As mentioned, the client SignalR Core library is no longer a jQuery plug-in. Put another way, if your UI is based on, say, Angular then you may not need to use jQuery at all.

Server-Side Events

The missing piece is the part of the solution that decides when it's about time to invoke a client function. There are two main scenarios. One is when the client invokes a server action through a Web API or a controller endpoint. The other is when the client directly invokes the hub. In the end, it's all about where you code the task that calls the client back.

In the canonical chat example, it all happens within the hub: performance of any required logic and dispatch of messages to the appropriate connection. Monitoring a remote task is a different thing. It assumes there's some business process running in the background that notifies its progress in some way. Technically, you might have this process coded in the hub and set up from there a conversation with the client UI. Or you can have the process triggered by a controller (API) and use the hub only as a way to pass events to the client. Even more realistically than shown in the example, you have the process coded in a layer below the level of controllers.

In a nutshell, you define the hub class and make it available wherever you can decide when and if to invoke client functions. The interesting thing is what's required to inject the hub instance into a controller or another business class. The demo injects the hub in a controller but it would do exactly the same for another deeper-level class. The sample TaskController is invoked via JavaScript directly from the client to trigger the remote task whose progresses will then move the progress bar:

```
public class TaskController : Controller
{
  private readonly IHubContext<ProgressHub> _progressHubContext;

  public TaskController(IHubContext<ProgressHub> progressHubContext)
  {
    _progressHubContext = progressHubContext;
  }
  public IActionResult Lengthy()
  {
    // Perform the task and call back
  }
}
```

You inject a hub in a controller or in any other class via the IHubContext<THub> interface. The IHubContext interface wraps up the hub instance but doesn't give you direct access to it. From there, you can dispatch messages back to the UI but you can't access, for example, the connection ID. Let's say the remote task is performed in the Lengthy method and it's from there you want to update the client progress bar:

```
progressHubContext
  .Clients
  .Client(connId)
  .InvokeAsync("updateProgressBar", 45);
```

The connection ID can be retrieved from within the hub class but not from within a generic hub context as in this example. Hence, the simplest way is for the client method to pass the connection string when it starts the remote task:

```
public void Lengthy([Bind(Prefix="id")] string connId) { … }
```

In the end, the controller class receives the SignalR connection ID, is injected the hub context and acts via the context calling methods through a generic non-typed API—the InvokeAsync method. Used in this way, there's no need to have methods in the hub class! If you find it surprising, well, take a look at the code at bit.ly/2DWd8SV.

Wrapping Up

This article discussed using ASP.NET Core SignalR in the context of a Web application to monitor a remote task. The hub is nearly empty as all the notification logic is built into the controller and orchestrated via the hub context injected via DI. This is just the starting point of a longer tour of ASP.NET Core SignalR. Next up, I'll delve deeper into the infrastructure and explore typed hubs, as well. ■

Dino Esposito *has authored more than 20 books and 1,000 articles in his 25-year career. Author of "The Sabbatical Break," a theatrical-style show, Esposito is busy writing software for a greener world as the digital strategist at BaxEnergy. Follow him on Twitter: @despos.*

Thanks to the following Microsoft expert for reviewing this article: *Andrew Stanton-Nurse*

Spreadsheets Everywhere.

SpreadsheetGear 2017 Released

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

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

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

Scalable Reporting

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

WIN Windows Forms | Silverlight | WPF

Powerful Controls

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

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.

Fastest Calculations

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

August 13 – 17, 2018
Redmond, WA

Microsoft Headquarters

Yesterday's Knowledge; Tomorrow's Code!

Visual Studio Live! (VSLive!™) is celebrating 25 years of coding innovation in 2018! From August 13 – 17, developers, software architects, engineers, designers and more will come together at Microsoft Headquarters for 5 days of unbiased education on the Microsoft Platform. Hone your skills in Visual Studio, ASP.NET Core, AngularJS, SQL Server, and so much more. Plus, you can eat lunch with the Blue Badges, rub elbows with Microsoft insiders, explore the campus, all while expanding your ability to create better apps!

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION

VSLive! 1997

VSLive! 2017

EVENT SPONSOR: Microsoft

PLATINUM SPONSOR: SMARTBEAR

SILVER SPONSOR: Progress

SUPPORTED BY: Visual Studio, msdn magazine, Visual Studio MAGAZINE

PRODUCED BY: 1105 MEDIA — YOUR GROWTH. OUR BUSINESS.

DEVELOPMENT TOPICS INCLUDE:

- Visual Studio / .NET
- Xamarin
- Angular / JavaScript
- Azure / Cloud
- Software Practices
- ASP.NET / Web Server
- ALM / DevOps
- Database & Analytics
- UWP (Windows)

"I return to Visual Studio Live! year after year (this is my third!) to see not only what bleeding edge technologies are being introduced, but to also see which technologies are fading; this helps me steer my organization in the right direction for our development needs. So many ideas gleaned from VSLive!, from Web APIs talking to SPA clients, to .NET Core, have helped me greatly!"

– Dilshan Jesook, KPMG

"I am going to look into using ASP.NET Core as the platform for new projects as a result of attending my first VSLive! event. I also think it would be easy to move some existing projects to it. I look forward to trying the new tools and creating just what I need!"

– Abby Quick, Epiq

Register to join us today!

Register Now and Save $400!

Use promo code VSLRED2

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/redmond

JAMES MCCAFFREY

Understanding LSTM Cells Using C#

A long short-term memory (LSTM) cell is a small software component that can be used to create a recurrent neural network that can make predictions relating to sequences of data. LSTM networks have been responsible for major breakthroughs in several areas of machine learning. In this article, I demonstrate how to implement an LSTM cell using C#. Although it's unlikely you'll ever need to create a recurrent neural network from scratch, understanding exactly how LSTM cells work will help you if you ever need to create an LSTM network using a code library such as Microsoft CNTK or Google TensorFlow.

The screenshot in **Figure 1** shows where this article is headed. The demo program creates an LSTM cell that accepts an input vector of size n = 2, and generates an explicit output vector of size m = 3 and a cell state vector of size m = 3. A key characteristic of LSTM cells is that they maintain a state.

An LSTM cell has (4 * n * m) + (4 * m * m) weights and (4 * m) biases. Weights and biases are just constants, with values like 0.1234, that define the behavior of the LSTM cell. The demo has 60 weights and 12 biases that are set to arbitrary values.

The demo sends input (1.0, 2.0) to the LSTM cell. The computed output is (0.0629, 0.0878, 0.1143) and the new cell state is (0.1143, 0.1554, 0.1973). I'll explain what these numeric values might represent shortly, but for now the point is that an LSTM cell accepts input, produces output and updates its state. The values of the weights and biases haven't changed.

Next, the demo feeds (3.0, 4.0) to the LSTM cell. The new computed output is (0.1282, 0.2066, 0.2883). The new cell state is (0.2278, 0.3523, 0.4789). Although it's not apparent, the new cell state value contains information about all previous input and output values. This is the "long" part of long short-term memory.

This article assumes you have intermediate or better programming skills, but doesn't assume you know anything about LSTM cells. The demo is coded using C#, but you should be able to refactor the code to another language, such as Python or Visual Basic, if you wish. The demo code is just a bit too long to present in its entirety, but the complete code is available in the accompanying file download.

Code download available at msdn.com/magazine/0418magcode.

Understanding LSTM Cells

There are three ways to describe LSTM cells: using an architecture diagram as shown in **Figure 2**; using math equations as shown in **Figure 3**; and using code as presented in this article. Your initial impression of the architecture diagram in **Figure 2** is probably something along the lines of, "What the heck?" The key ideas are that an LSTM cell accepts an input vector x(t), where the t stands for a time. The explicit output is h(t). The unusual use of h (rather than o) to represent output is historical and comes from the fact that neural systems were often described as functions g and h.

The LSTM cell also uses h(t-1) and c(t-1) as inputs. Here, the t-1 means from the previous time step. The c represents the cell state, so h(t-1) and c(t-1) are the previous output values and the previous state values. The interior of the LSTM cell architecture looks complicated. And it is, but not as complicated as you might guess.

```
file:///C:/LSTM_IO/bin/Debug/LSTM_IO.EXE

Begin LSTM IO demo

Creating an n=2 input, m=3 state LSTM cell
Setting LSTM weights and biases to small arbitrary values

Sending input = (1.0, 2.0) to LSTM

Output is:
0.0629
0.0878
0.1143

New cell state is:
0.1143
0.1554
0.1973

=====

Sending input = (3.0, 4.0) to LSTM

Output is:
0.1282
0.2066
0.2883

New cell state is:
0.2278
0.3523
0.4789

End LSTM demo
```

Figure 1 **LSTM Cell Input-Output Demo**

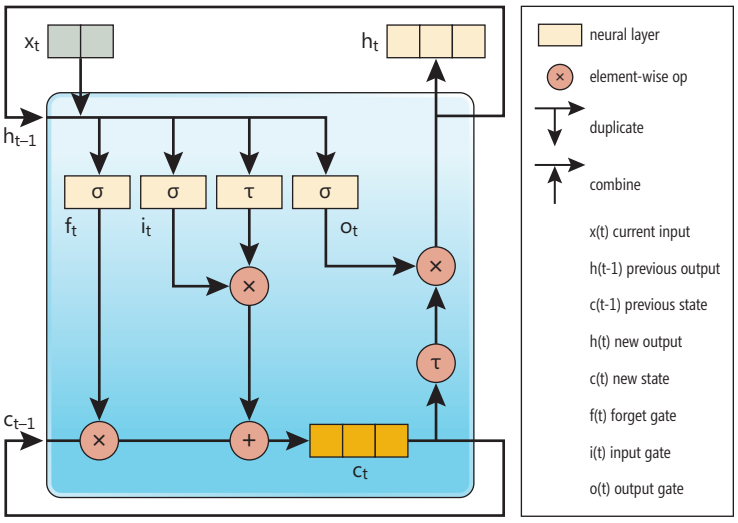


Figure 2 **LSTM Cell Architecture**

The math equations in **Figure 3** define the behavior of the demo program LSTM cell. If you don't regularly work with math definitions, your reaction to **Figure 3** is likely, again, "What the heck?" Equations (1), (2) and (3) define three gates: a forget gate, an input gate and an output gate. Each gate is a vector of values between 0.0 and 1.0, which are used to determine how much information to forget (or, equivalently, remember) in each input-output cycle. Equation (4) computes the new cell state, and equation (5) computes the new output.

These equations are simpler than they appear. For example, equation (1) is implemented by the demo code as:

```
float[][] ft = MatSig(MatSum(MatProd(Wf, xt),
                MatProd(Uf, h_prev), bf));
```

Here, MatSig is a program-defined function that applies logistic-sigmoid to each value in a matrix. MatSum adds three matrices. MatProd multiplies two matrices. Once you understand basic matrix and vector operations, implementing an LSTM cell is quite easy.

Overall Demo Program Structure

The structure of the demo program, with a few minor edits to save space, is presented in **Figure 4**. The demo uses a static method approach rather than an OOP approach to keep the main ideas as clear as possible.

To create the demo, I launched Visual Studio and created a new C# console application named LSTM_IO. I used Visual Studio 2015, but the demo has no significant .NET Framework dependencies, so any version of Visual Studio will work fine.

After the template code loaded, in the Solution Explorer window I renamed file Program.cs to LSTM_IO_Program.cs and allowed Visual Studio to automatically rename class Program for me. At the top of the editor window, I deleted all unneeded references to namespaces, leaving just the one to the top-level System namespace. All the work is performed by function ComputeOutputs.

Matrices Using C#

In order to implement an LSTM cell, you must have a solid grasp of working with C# matrices. In C#, a matrix is an array-of-arrays. In machine learning, it's common to use 32-bit type float rather than 64-bit byte double.

The demo defines a helper to create a matrix as:

```
static float[][] MatCreate(int rows, int cols)
{
  float[][] result = new float[rows][];
  for (int i = 0; i < rows; ++i)
    result[i] = new float[cols];
  return result;
}
```

The first statement creates an array with the specified number of rows, where each row is an array of type float. The for-loop statement allocates each row as an array with the specified number of columns. Note that unlike most programming languages, C# supports a true matrix type, but using an array-of-array approach is much more common.

In machine learning, the term *column vector*, or just vector for short, refers to a matrix with one column. Most machine leaning code works with vectors rather than one-dimensional arrays. The demo defines a function to generate a matrix/vector from a one-dimensional array:

```
static float[][] MatFromArray(float[] arr, int rows, int cols)
{
  float[][] result = MatCreate(rows, cols);
  int k = 0;
  for (int i = 0; i < rows; ++i)
    for (int j = 0; j < cols; ++j)
      result[i][j] = arr[k++];
  return result;
}
```

The function can be called to create a 3x1 (3 rows, 1 column) vector like this:

```
float[][] v = MatFromArray(new float[] {1.0f, 9.0f, 5.0f}, 3,1);
```

(1) $f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$

(2) $i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$

(3) $o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$

(4) $c_t = f_t \circ c_{t-1} + i_t \circ \tau(W_c x_t + U_c h_{t-1} + b_c)$

(5) $h_t = o_t \circ \tau(c_t)$

n : size of input
m : size of cell state and output
x_t : input vector, time t, size n×1
f_t : forget gate vector, size m×1
i_t : input gate vector, size m×1
o_t : output gate vector, size m×1
h_t : output vector, size m×1
c_t : cell state vector, size m×1
W_f, W_i, W_o, W_c : input gate weight matrices, size m×n
U_f, U_i, U_o, U_c : output gate weight matrices, size m×m
b_f, b_i, b_o, b_c : bias vectors, size m×1
σ : logistic sigmoid activation function
τ : tanh activation function

Figure 3 **LSTM Cell Math Equations**

The demo defines a MatCopy function that can duplicate a matrix. MatCopy can be called:

```
float[][] B = MatCopy(A);
```

Here, B is a new, independent matrix with the same values as A. Be careful of code like:

```
float[][] B = A;
```

This creates B as a reference to A, so any change made to either matrix affects the other. This may be the behavior you want, but probably not.

Matrix Element-Wise Operations

An LSTM cell implementation uses several element-wise functions on matrices, where each value in a matrix is used or modified. For example, function MatTanh is defined:

```
static float[][] MatTanh(float[][] m)
{
  int rows = m.Length; int cols = m[0].Length;
  float[][] result = MatCreate(rows, cols);
  for (int i = 0; i < rows; ++i) // Each row
    for (int j = 0; j < cols; ++j) // Each col
      result[i][j] = Tanh(m[i][j]);
  return result;
}
```

The function traverses its input matrix m and applies the hyperbolic tangent (tanh) to each value. Helper function Tanh is defined:

```
static float Tanh(float x)
{
  if (x < -10.0) return -1.0f;
  else if (x > 10.0) return 1.0f;
  return (float)(Math.Tanh(x));
}
```

Figure 4 **Demo Program Structure**

```
using System;
namespace LSTM_IO
{
  class LSTM_IO_Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin LSTM IO demo");

      // Set up inputs
      // Set up weights and biases

      float[][] ht, ct;  // Outputs, new state
      float[][][] result;

      result = ComputeOutputs(xt, h_prev, c_prev,
        Wf, Wi, Wo, Wc, Uf, Ui, Uo, Uc, bf, bi, bo, bc);

      ht = result[0];  // Outputs
      ct = result[1];  // New state

      Console.WriteLine("Output is:");
      MatPrint(ht, 4, true);
      Console.WriteLine("New cell state is:");
      MatPrint(ct, 4, true);

      // Set up new inputs
      // Call ComputeOutputs again

      Console.WriteLine("End LSTM demo");
    }

    static float[][][] ComputeOutputs(float[][] xt,
      float[][] h_prev, float[][] c_prev,
      float[][] Wf, float[][] Wi, float[][] Wo, float[][] Wc,
      float[][] Uf, float[][] Ui, float[][] Uo, float[][] Uc,
      float[][] bf, float[][] bi, float[][] bo, float[][] bc)
    { . . }

    // Helper matrix functions defined here
  }
}
```

The demo also defines a MatSigmoid function that's exactly like MatTanh except logistic-sigmoid is applied to each value. The logistic-sigmoid function is closely related to tanh and returns a value between 0.0 and 1.0 instead of between -1.0 and +1.0.

The demo defines a function MatSum that adds the values in two matrices of the same shape. If you look at math equation (1) in **Figure 3**, you'll see that an LSTM adds three matrices. The demo overloads MatSum to work with two or three matrices.

Function MatHada multiplies corresponding values in two matrices that have the same shape:

```
static float[][] MatHada(float[][] a, float[][] b)
{
  int rows = a.Length; int cols = a[0].Length;
  float[][] result = MatCreate(rows, cols);
  for (int i = 0; i < rows; ++i)
    for (int j = 0; j < cols; ++j)
      result[i][j] = a[i][j] * b[i][j];
  return result;
}
```

Element-wise multiplication is sometimes called the Hadamard function. In the math equations (4) and (5) in **Figure 3**, the Hadamard function is indicated by the open dot symbol. Don't confuse the element-wise Hadamard function with matrix multiplication, which is a very different function.

Matrix Multiplication

If you haven't seen matrix multiplication before, the operation is not at all obvious. Suppose A is a 3x2 matrix:

```
1.0, 2.0
3.0, 4.0
5.0, 6.0
```

And suppose B is a 2x4 matrix:

```
10.0, 11.0, 12.0, 13.0
14.0, 15.0, 16.0, 17.0
```

The result of C = AB (multiplying A and B) is a 3x4 matrix:

```
 38.0  41.0  44.0  47.0
 86.0  93.0 100.0 107.0
134.0 145.0 156.0 167.0
```

The demo implements matrix multiplication as function MatProd. Note that when using C#, for very large matrices you can use the Parallel.For statement in the Task Parallel Library.

To summarize, implementing an LSTM cell using C# requires several helper functions that create and operate on matrices (arrays-of-arrays) and vectors (matrices with one column). The demo code defines functions MatCreate, MatFromArray, MatCopy(m), MatSig(m), MatTanh(m), MatHada(a, b), MatSum(a, b), MatSum(a, b, c) and MatProd(a, b). Although not essential for creating an LSTM cell, it's useful to have a function to display a C# matrix. The demo defines at MatPrint function.

Implementing and Calling LSTM Input-Output

The code for function ComputeOutputs is presented in **Figure 5**. The function has 15 parameters, but 12 of them are essentially the same.

Vector xt is the input, such as (1.0, 2.0). Vectors h_prev and c_prev are the previous output vector and previous cell state. The four W matrices are the gate weights associated with input values, where f is a forget gate, i is an input gate and o is an output gate. The four U matrices are the weights associated with cell output. The four b vectors are biases.

In the body of the function, the first five statements are a one-to-one mapping to the five math equations in **Figure 3**. Notice that the ft, it and ot gates all use the MatSig function. Therefore, all three are vectors with values between 0.0 and 1.0. You can think of these as filters that are applied to input, output or state, where the value in the gate is the percentage retained. For example, if one of the values in ft is 0.75, then 75 percent of the corresponding value in the combined input and previous-output vector is retained. Or, equivalently, 25 percent of the information is forgotten.

The computation of the new cell state, ct, is simple to implement but conceptually quite deep. At a high level, the new cell state depends on a gated combination of the input vector xt, and the previous output vector and cell state, h_prev and c_prev. The new output, ht, depends on the new cell state and the output gate. Quite remarkable.

The function returns the new output and new cell state in an array. This leads to a return type of float[][][], where result[0] is an array-of-arrays matrix holding the output and result[1] holds the new cell state.

Calling ComputeOutputs is mostly a matter of setting up the parameter values. The demo begins the preparation with:

```
float[][] xt = MatFromArray(new float[] {
  1.0f, 2.0f }, 2, 1);
float[][] h_prev = MatFromArray(new float[] {
  0.0f, 0.0f, 0.0f }, 3, 1);
float[][] c_prev = MatFromArray(new float[] {
  0.0f, 0.0f, 0.0f }, 3, 1);
```

Both the previous output and the cell state are explicitly initialized to zero. Next, two sets of arbitrary weight values are created:

```
float[][] W = MatFromArray(new float[] {
  0.01f, 0.02f,
  0.03f, 0.04f,
  0.05f, 0.06f }, 3, 2);

float[][] U = MatFromArray(new float[] {
  0.07f, 0.08f, 0.09f,
  0.10f, 0.11f, 0.12f,
  0.13f, 0.14f, 0.15f }, 3, 3);
```

Notice that the two matrices have different shapes. The weight values are copied to the input parameters:

```
float[][] Wf = MatCopy(W); float[][] Wi = MatCopy(W);
float[][] Wo = MatCopy(W); float[][] Wc = MatCopy(W);
float[][] Uf = MatCopy(U); float[][] Ui = MatCopy(U);
float[][] Uo = MatCopy(U); float[][] Uc = MatCopy(U);
```

Figure 5 **Function ComputeOutputs**

```
static float[][][] ComputeOutputs(float[][] xt,
  float[][] h_prev, float[][] c_prev,
  float[][] Wf, float[][] Wi, float[][] Wo, float[][] Wc,
  float[][] Uf, float[][] Ui, float[][] Uo, float[][] Uc,
  float[][] bf, float[][] bi, float[][] bo, float[][] bc)
{
  float[][] ft = MatSig(MatSum(MatProd(Wf, xt),
    MatProd(Uf, h_prev), bf));
  float[][] it = MatSig(MatSum(MatProd(Wi, xt),
    MatProd(Ui, h_prev), bi));
  float[][] ot = MatSig(MatSum(MatProd(Wo, xt),
    MatProd(Uo, h_prev), bo));
  float[][] ct = MatSum(MatHada(ft, c_prev),
    MatHada(it, MatTanh(MatSum(MatProd(Wc, xt),
      MatProd(Uc, h_prev), bc))));
  float[][] ht = MatHada(ot, MatTanh(ct));

  float[][][] result = new float[2][][];
  result[0] = MatCopy(ht);
  result[1] = MatCopy(ct);
  return result;
}
```

Because the weights don't change, the demo could have assigned by reference instead of using MatCopy. The biases are set up using the same pattern:

```
float[][] b = MatFromArray(new float[] {
  0.16f, 0.17f, 0.18f }, 3, 1);
float[][] bf = MatCopy(b); float[][] bi = MatCopy(b);
float[][] bo = MatCopy(b); float[][] bc = MatCopy(b);
```

Function ComputeOutputs is called like this:

```
float[][] ht, ct;
float[][][] result;
result = ComputeOutputs(xt, h_prev, c_prev,
  Wf, Wi, Wo, Wc, Uf, Ui, Uo, Uc,
  bf, bi, bo, bc);
ht = result[0];  // Output
ct = result[1];  // New cell state
```

The whole point of an LSTM cell is to feed a sequence of input vectors, so the demo sets up and sends a second input vector:

```
h_prev = MatCopy(ht);
c_prev = MatCopy(ct);
xt = MatFromArray(new float[] {
  3.0f, 4.0f }, 2, 1);
result = ComputeOutputs(xt, h_prev, c_prev,
  Wf, Wi, Wo, Wc, Uf, Ui, Uo, Uc,
  bf, bi, bo, bc);
ht = result[0];
ct = result[1];
```

Note that the demo explicitly sends the previous output and state vectors to ComputeOutputs. An alternative is to feed just the new input vector, because the previous output and cell state are still stored in ht and ct.

Connecting the Dots

So, what's the point? An LSTM cell can be used to construct an LSTM recurrent neural network—an LSTM cell with some additional plumbing. These networks have been responsible for major advances in prediction systems that work with sequence data. For example, suppose you were asked to predict the next word in the sentence, "In 2017, the championship was won by ___." With just that information, you'd be hard pressed to make a prediction. But suppose your system had state, and remembered that part of a previous sentence was, "The NBA has held a championship game since 1947." You'd now be in a position to predict one of the 30 NBA teams.

There are dozens of variations of LSTM architectures. Additionally, because LSTM cells are complex, there are dozens of implementation variations for every architecture. But if you understand the basic LSTM cell mechanism, you can easily understand the variations.

The demo program sets the LSTM weights and biases to arbitrary values. The weights and biases for a real-world LSTM network would be determined by training the network. You'd obtain a set of training data with known input values and known, correct output values. Then you'd use an algorithm such as back-propagation to find values for the weights and biases that minimize error between computed outputs and correct outputs. ■

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

Thanks to the following Microsoft technical experts who reviewed this article: *Ricky Loynd and Adith Swaminathan*

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

September 17-20, 2018
Renaissance
Chicago

Look Back to Code Forward

Visual Studio Live! (VSLive!™) is thrilled to be returning to Chicago where developers, software architects, engineers and designers will "look back to code forward" during four days of unbiased and cutting-edge education on the Microsoft Platform. Tackle training on the hottest topics (like .NET Core, Angular, VS2017), debate with industry and Microsoft insiders (people like Rockford Lhotka, Deborah Kurata and Brock Allen) and network with your peers—plus, help us celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Come experience the education, knowledge-share and networking at #VSLive25.

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION 1993 - 2018

VSLive! 1998


VSLive! 2016

SILVER SPONSOR

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA — YOUR GROWTH, OUR BUSINESS.

DEVELOPMENT TOPICS INCLUDE:

- Visual Studio / .NET
- JavaScript / Angular
- Xamarin
- Software Practices
- Database & Analytics
- ASP.NET / Web Server
- ALM / DevOps
- Azure / Cloud
- UWP

Who Should Attend and Why

We've been around since 1993. What's our secret? YOU! Since our first conference (VBITS/VSLive!/Visual Studio Live!), tens of thousands of developers, software architects, programmers, engineers, designers and more have trusted us year-in-and-year-out for unbiased and cutting-edge education on the Microsoft Platform.

@ vslive.com/chicagomsdn

Register to join us today.

Register Now and Save Up To $400!

Use promo code VSLCH2

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/chicagomsdn

DAVID S. PLATT

I've Got a Little List

"When are you going to spoof another song?" my readers keep asking, referring to my ruination of Billy Joel's "We Didn't Start the Fire" for my 2014 April Fool's Day column (msdn.com/magazine/dn630653). I can't tell if they're asking to ensure they don't miss it, or ensure they avoid it.

For this April Fools' Day, I've decided to spoil Gilbert and Sullivan's 1885 comic opera, "The Mikado" (bit.ly/2oIoGp5), which gives us today's word "poohbah," an eponym from the character designated "Lord High Everything Else." Ostensibly set in Japan, it allowed Gilbert and Sullivan (not Gilbert O'Sullivan, bit.ly/2C6ImWK, you philistines) to satirize British institutions and politics without incurring the wrath a direct attack would have unleashed—as I have to do with Microsoft sometimes.

I've wrecked the song "As Some Day It May Happen," better known as "I've Got a Little List," sung by Ko-Ko, the Lord High Executioner (straight version at youtu.be/CyZu7mQt18Q, other spoofs at youtu.be/CWo_3ClcTBQ and bit.ly/2Cv1nOn and youtu.be/WIObOXMxt84 and youtu.be/VXfkCzcegiM.) He's got a little list, of things and people the world would be better off without. We all have these lists, don't we, my friends? On how many of yours do I appear?

If Ko-Ko had to struggle with today's software, I'm sure his list would contain many more entries. He'd probably sing something like this:

As some day it may happen that a victim must be found
I've got a little list — I've got a little list
Of our software biz offenders who might well be underground
And who never would be missed — who never would be missed!

All those confirmation boxes always asking "Are you sure?"
And the daily breaking updates and their crashes we endure
Guys who sing while wearing earbuds, who can't hear how bad they sound
Login passwords that change monthly, so you have to write them down
Sites that promise a free trial but on a credit card insist
They'd none of 'em be missed - they'd none of 'em be missed!

There's the doc who turns his back on you to type on his PC
And that budget analyst — I've got him on the list!
Facebook pictures of your breakfast. Do we really need to see?
They never would be missed — they never would be missed!

Those grammatical assistants always popping up to yelp,
"I see you wrote a parody; a bad one. Can I help?"
And idiots with selfie sticks that bash you in the head
Taking pictures petting tigers. Are they trying to get dead?
They all drive you up a wall until you need a therapist
I don't think they'd be missed — I'm sure they'd not be missed!

Tech support that takes an hour, and then asks, "Are you plugged in?"
IT killjoys who block sites at work to shield you from a sin
PowerPointers droning on through every bullet point they read
A whole bag of different dongles, all except the one you need
Guys who show up late for meetings and then say, "What did I miss?"
They never would be missed — they never would be missed!

I've tackled the song "As Some Day It May Happen," better known as "I've Got a Little List," sung by Ko-Ko, the Lord High Executioner. He's got a little list, of things and people the world would be better off without.

There's the splash screen that won't go away until you wait awhile
And that Plattski satirist — I've got him on the list!
Social media echo chambers that just reinforce your bile
They'd none of 'em be missed — they'd none of 'em be missed
And Nigerian oil princes of a spam producing kind
Such as — What d'ye call him — Thingamabob and, likewise, Never mind

And all those self-igniting batteries, and also You-know-who
The task of filling up the blanks I'd rather leave to you
There're so many things and people that belong upon this list
For they'd none of 'em be missed — they'd none of 'em be missed! ■

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.*

STEAM UNIVERSE
THE WAY FORWARD FOR EDUCATION

IN PARTNERSHIP WITH

THE JOURNAL

CAMPUS TECHNOLOGY

YOUR NEW GO-TO RESOURCE

STEAM FOR EDUCATION

Artificial Intelligence

Makerspaces

Computer Science

Grants

Professional Development

3D Printers

Internet of Things

And MORE!

STEAMUniverse.com