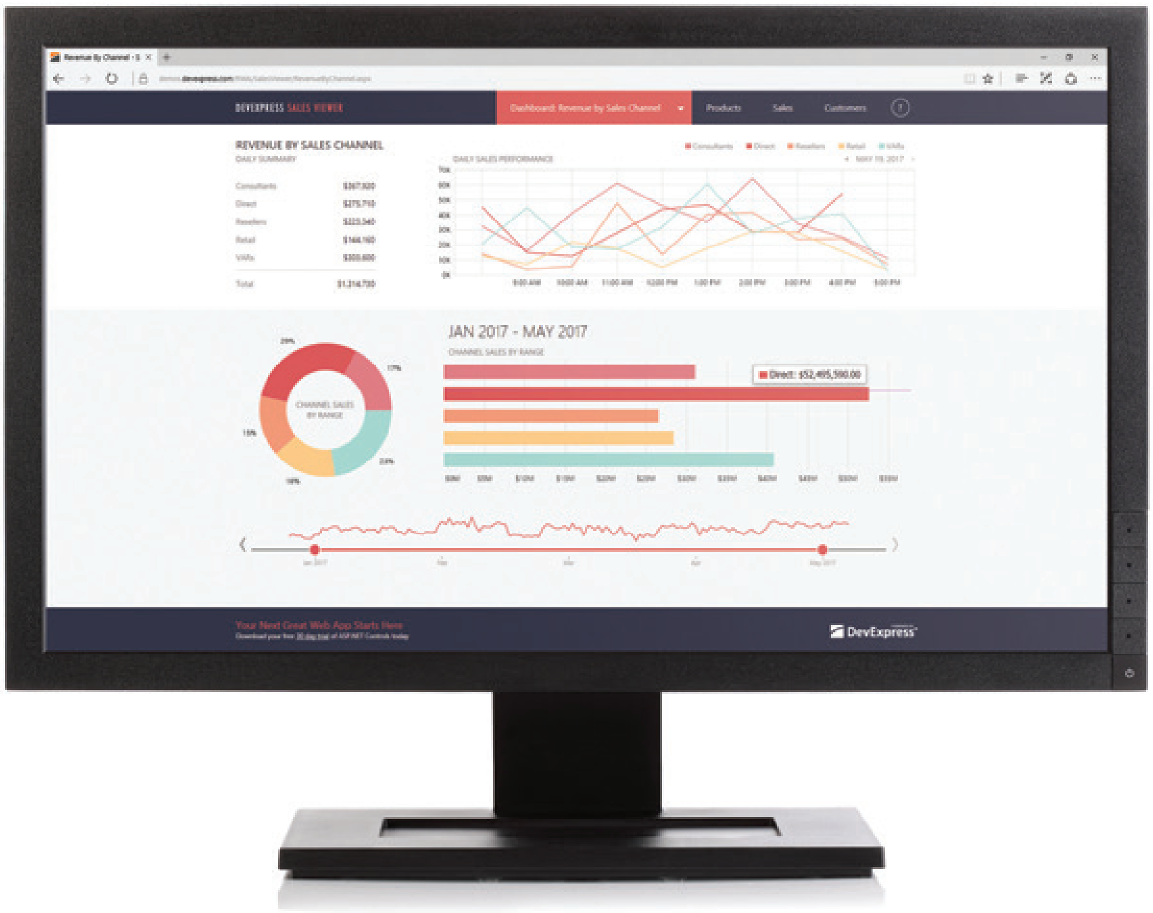
THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2017 VOL 32 NO 12

msdn magazine



THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2017 VOL 32 NO 12

msdn magazine

COLUMNS

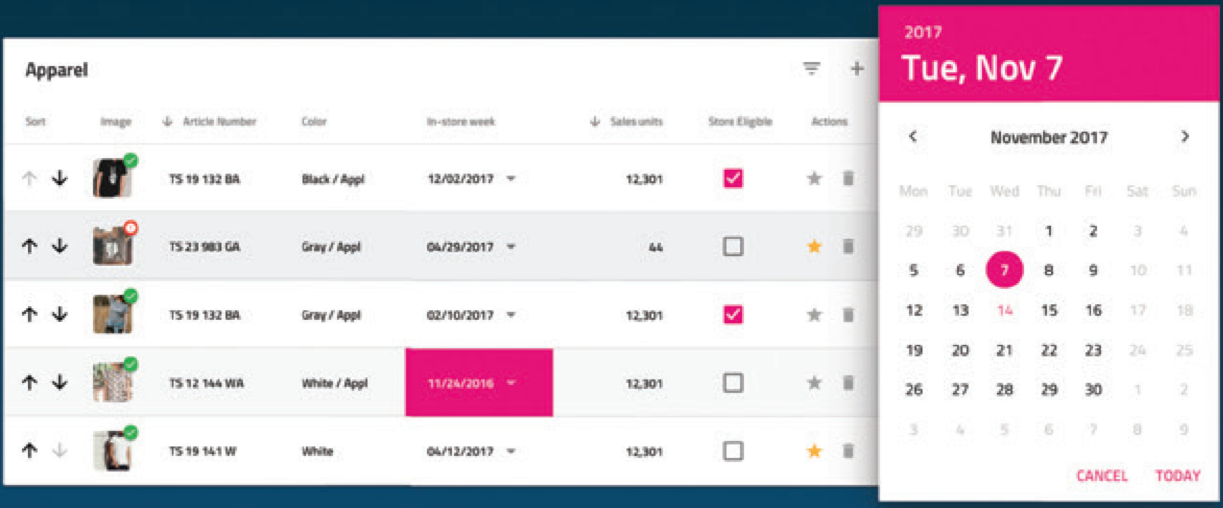
Microsoft

-0.34 (-0.72%)
MAY
APRIL
MARCH
High
Low
Open
Volume
Clear Filter
Starts with
Ends with
Contains
Does not contain
Equals
Does not equal
4:30 PM
Carrier
MOO2U
Cupcake Factory
Tom Washington
Little Richie Software
WPF Disciples
Abel Fox
Le Chocolateur
25%
50%
0%
$234,098
12%
CAMPAIGN HEALTH
TOTAL
200K
20%
TRAFFIC BY REGION

Featuring

Ignite UI

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



Apparel

Sort	Image	↓ Article Number	Color	In-store week	↓ Sales units	Store Eligible	Actions
↑ ↓		TS 19 132 BA	Black / Appl	12/02/2017 ▾	12,301	☑	★ 🗑
↑ ↓		TS 23 983 GA	Gray / Appl	04/29/2017 ▾	44	☐	★ 🗑
↑ ↓		TS 19 132 BA	Gray / Appl	02/10/2017 ▾	12,301	☑	★ 🗑
↑ ↓		TS 12 144 WA	White / Appl	11/24/2016 ▾	12,301	☐	★ 🗑
↑ ↓		TS 19 141 W	White	04/12/2017 ▾	12,301	☐	★ 🗑

2017
Tue, Nov 7

‹ November 2017 ›

Mon	Tue	Wed	Thu	Fri	Sat	Sun
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

- ✓ 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

DECEMBER 2017 VOLUME 32 NUMBER 12

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

ENTERPRISE COMPUTING GROUP

President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

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

PRODUCTION STAFF
Print Production Manager **Peter B. Weller**
Print Production Coordinator **Lee Alexander**

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

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

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

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

1105 MEDIA INC
YOUR GROWTH. OUR BUSINESS.

Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Chief Financial Officer
Craig Rucker

Chief Technology Officer
Erik A. Lindgren

Executive Vice President
Michael J. Valenti

Chairman of the Board
Jeffrey S. Klein

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

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

EDITOR'S NOTE

MICHAEL DESMOND

Hamburger Helper

When the Windows XAML team at Microsoft developed the new NavigationView control as part of the Windows 10 Fall Creators Update, it addressed an important gap in the Universal Windows Platform (UWP) UI. As Jerry Nixon writes in his feature this month, "A Developer's Guide to the New Hamburger Menu in Windows 10," internal Microsoft development teams were looking for ways to improve visual alignment and consistency across their portfolio of apps—something that proved difficult to do with the existing SplitView control.

> Nixon also singles out useful new capabilities like Conditional XAML, which, starting with the Windows 10 Fall Creators Update, allows developers to set properties and instantiate objects at run time based on the presence of a detected API.

"The internal teams were creating their own hamburger implementations, but they needed to ensure consistent user experience across modalities like pen, ink and touch," Nixon explains. "They also had very specific requirements around accessibility that were difficult to achieve."

The NavigationView control helped resolve those challenges, enabling the teams to fine-tune app UXes to meet specific requirements, while providing all the benefits of an in-box control. As Nixon notes, Microsoft-built controls meet "stringent performance, design, localization, support and accessibility requirements that most third-party and open source projects do not."

Even as Microsoft improves the state of the art in its UWP XAML controls, the company continues work to unify XAML dialects like UWP XAML and Xamarin.Forms around a single target. The XAML Standard specification defines a common XAML vocabulary that enables supporting frameworks to share XAML-based UI definitions. It's a tricky business that requires balancing the interests of incumbent Windows Presentation Foundation (WPF) and Silverlight developers while winning over UWP and Xamarin programmers. The payoff for those last two audiences, Nixon says, could be big.

"Xamarin is proposing that with XAML Standard they can transpile a UI design into many different platforms, including UWP. This means a developer can write once and get many out of it, but the resulting app for UWP is still UWP XAML transpiled from XAML Standard," Nixon explains. "When a developer wants to enhance an app on Android or Windows, it takes those specific skills to be successful. So, it does not eclipse UWP as much as it accelerates it."

Nixon also singles out useful new capabilities like Conditional XAML, which, starting with the Windows 10 Fall Creators Update, allows developers to set properties and instantiate objects at run time based on the presence of a detected API. Conditional XAML lets developers fully optimize the UX while preserving backward compatibility.

As Nixon concludes near the end of his feature article, "No control meets the needs of every developer. No API meets the needs of every design." That's certainly true of XAML, which remains a platform in progress with a diverse constituency spanning multiple frameworks and device families. But with its continuing investments, both in tooling and controls and in the larger-scope XAML unification effort, Microsoft seems determined to expand the utility and appeal of XAML.

And it all begins with a hamburger.

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

Managing the Manager: 15 Tips for Working Better

Broadly speaking, engineering managers span the spectrum from micromanagers to completely hands-off bosses. The challenge for developers is understanding how to "manage up," no matter where their managers are on the spectrum. Here are 15 ideas that can help:

Hands-off Managers

1. Mimic. It's tempting to over-communicate with hands-off managers, sending them detailed status reports or describing everything you're up to. Unconsciously, you may want to "prove" to them that you're trustworthy and independent. Instead, be hands-off with them! For instance, loop them in only when you have high-impact news to share, or when you need their help.

2. Vindicate. With this approach, a hands-off manager is trusting you by default, until you give her reason to think otherwise. Fulfill this trust by being thorough, conscientious, detail-oriented and honest in your projects. The more you do this, the more freedom you earn.

3. Escalate. Sometimes, a hands-off approach may not be in the best interest of the team or project. In such a situation, it's OK to request more support and hand-holding. Most managers will happily oblige—in fact, they may have been waiting for you to ask!

4. Reinforce. If you like managers who are hands-off, tell them so, regularly, and explain why. This encourages them to relax even more.

5. Credit. Make a point to share credit with your hands-off manager, even if she plays only a passing role in a project. Identify her as a co-creator or co-author. This isn't empty praise. Your manager's limited involvement is often instrumental to your success and growth.

Micromanagers

1. Object. It can be important to set limits with an overly active manager. If you have a valid complaint, approach the conversation so that it's about how you feel, rather than what they did wrong. Focus on what you and your manager can do together to improve things. As an example, you could say, "When you did X, it made me feel Y. What can we do together to avoid this in the future?"

2. Prevent. Signs of trouble can be spotted before you ever take a job, such as in the way your future manager negotiates salary or title, deals with an offer deadline, or speaks about his colleagues or bosses. You can honor those red flags by not taking the role.

3. Tolerate. Anytime I've made a "dramatic exit" to make a point, by walking out on a micromanager, I've come to regret it. It weakens my bonds to *everyone* on that team and closes the door on future opportunities. If you do feel the need to leave the team, wait until you no longer feel furious.

4. Flex. You might disagree with a micromanager's approach to a project, but implementing her direction can actually help expand your worldview. By accepting her opinion over your personal judgment, you learn how to not be overly wedded to your beliefs—and occasionally, you may even start to see the validity of hers.

5. Silence. If your micromanager insults or railroads you, silence might be the best (and only!) viable response. Other times, you could say, "I don't know right now," or, "I feel overwhelmed, so let me think about it." More broadly, you could even reframe your entire relationship with your manager, viewing it as a vehicle for serving others and learning to accept those who are different.

All Managers

1. Lead. Most managers care about the quality of your work, and the results you achieve, regardless of their level of day-to-day involvement. So, in your chats with them, focus on *high-level outcomes*. You can also describe steps you're taking to ensure highest-quality results. This approach elevates your conversations to focus on *how* you work and what you achieve, rather than the nitty-gritty of daily bug fixes or implementation details.

2. Delimit. When you ask your manager for help, focus her attention by being *very* specific in your ask. This helps ground your manager's thinking to produce actionable advice.

3. Connect. Managers are human, so you can consciously deepen your personal relationship with them. If you genuinely care, you can ask them about their family, health, hobbies and the like. These conversations deepen your understanding of your manager, and the shared vulnerability from them can bring both of you closer together.

4. Schedule. It's good to schedule separate one-on-one time with managers devoted solely to the discussion of your career goals and performance. These focused discussions allow them to be *hands-on* in growing your responsibilities and career.

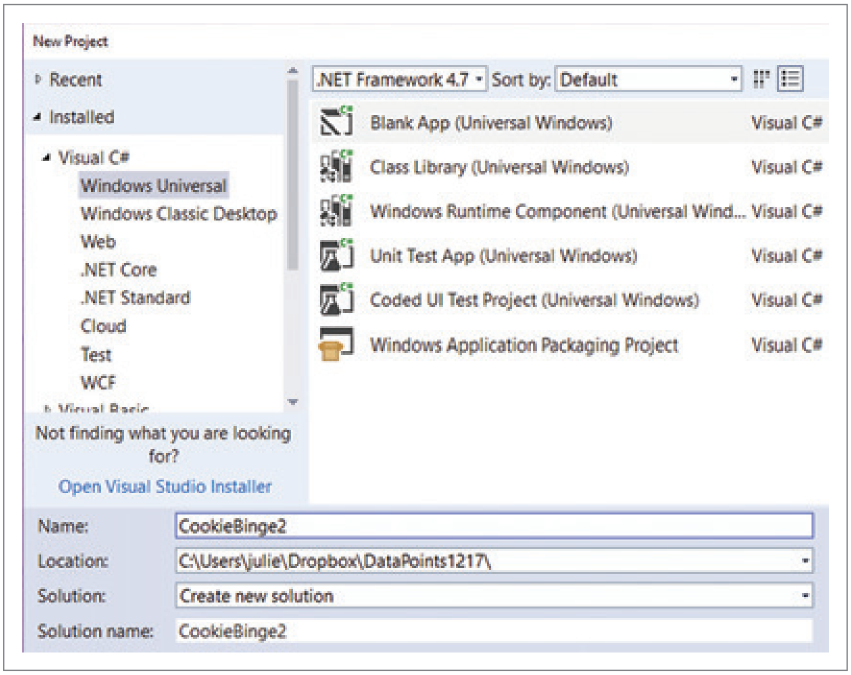
5. Clear. Identify your manager's top-of-mind priorities, then go to work on one of those. You'll not only accelerate your growth, you'll also train yourself to be more perceptive. ■

KRISHNAN RANGACHARI *is an advisor to CTOs, CIOs, and VPs of Engineering on organizational transformation, culture creation, and engineering leadership. Visit RadicalShifts.com for his private newsletter and webinars.*

FILE
HOME
INSERT
PAGE LAYOUT
AutoSum
Financial
Logical
Text
Date & Time
Lookup & Reference
Math & Trig
More
Function Library
Name Manager
Use in Formula
Create from Selection
Defined Names
Show Formulas
Formula Aud...
Options
Calculation
=SUM('Financial Data'!H31:H42)
ALL
$12,687,146.00
Televisions
$9,795,626.00
Monitors
$742,635.00
Video Players
$1,202,610.00
Automation
$946,275.00
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
FY 2011
FY 2012
FY 2013
% CHANGE
DesktopLCD 19
DesktopLCD 21
DesktopLED 19
DesktopLED 21
$0.00
$0.00
$0.00
$0.00
$104,550.00
$144,720.00
$154,500.00
$190,250.00
$126,550.00
$214,920.00
$154,040.00
$247,125.00
Financial Report
Financial Data

Building UWP Apps for Local and Cloud Data Storage

This is the first of a multi-part series that will show you how to store data on a device running a Universal Windows Platform (UWP) app, as well as how to store the app's data in the cloud. This article will focus on the local storage using Entity Framework Core (EF Core) and a SQLite database. The subsequent parts will add in capabilities to store and retrieve the UWP app data to the cloud using Azure Functions and Azure Cosmos DB.


New Project
Recent
Installed
Visual C#
Windows Universal
Windows Classic Desktop
Web
.NET Core
.NET Standard
Cloud
Test
WCF
Visual Basic
Not finding what you are looking for?
Open Visual Studio Installer
.NET Framework 4.7
Sort by: Default
Blank App (Universal Windows) Visual C#
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Wind... Visual C#
Unit Test App (Universal Windows) Visual C#
Coded UI Test Project (Universal Windows) Visual C#
Windows Application Packaging Project Visual C#

Name: CookieBinge2
Location: C:\Users\julie\Dropbox\DataPoints1217\
Solution: Create new solution
Solution name: CookieBinge2

Figure 1 **Creating a UWP Project from the Blank App Template**

In the early days of EF Core, when it was still called EF7, I took advantage of its new ability to run not just on a full .NET Framework setup, but on devices, as well. In my first Pluralsight course on EF7 (which was designed as a preview of the features being built), I created a small and quite silly game called Cookie Binge. It ran on Windows Phone 8 and as a Windows Store app for Windows 8, storing its data locally using EF7 and SQLite. The game was a spinoff of a demo app built by the EF team that focused on capturing unicorns. The CookieBinge game lets you eat cookies (by clicking on them) and when you're finished binging, you indicate either that the spree was totally worth it or that you feel guilty for scarfing down all those cookies. The number of cookies you consume becomes your score and your selection of "worth it" or "guilty" is tied to the score in the game's history. It's a silly game with no real goal, just my way of exploring how to incorporate EF 7/Core into a mobile app.

When EF Core was released in 2016, I evolved the game to run as a Universal Windows Platform (UWP) app, which could be played on any device running Windows 10. The recently released EF Core 2.0 now has a dependency on .NET Standard 2.0. The latest version of UWP, which targets the just-released Windows 10 Fall Creators Update, also relies on .NET Standard 2.0, allowing you to use EF Core 2.0 in this new generation of UWP apps.

That means it's time for me to update the CookieBinge app yet again, this time to EF Core 2.0. The only way to do that is to also update the UWP app to target the new version of Windows 10. Keep in mind that my focus will remain on the data access, not on designing the UI of the app to benefit from the features of Windows 10. Additionally, I won't cover the new features of EF Core 2.0 in this example. Instead, I'll be looking at the ability to combine these two technologies.

Code download available at msdn.com/magazine/1217magcode.

I'll build the application from scratch, rather than updating the existing app, so you can follow along. After I get the game updated to EF Core 2.0, I'll add a new feature to allow sharing of scores with other game players. To accomplish this, in addition to storing scores locally on the game device, the app will leverage Azure Functions and Azure Cosmos DB so users can share their scores around the globe. I'll perform these tasks in subsequent installments of this series.

Your Dev Environment

Although EF Core can be built and run on cross-platform environments, UWP is a Windows-only platform, so you'll need to do this work on a Windows machine. To be specific, you'll need Windows 10 Fall Creators Update and Visual Studio 2017 version 15.4 (or later) installed. Visual Studio doesn't install the .NET Standard 2.0 SDK, so you must install that manually. You'll find links to the .NET Standard downloads at bit.ly/2mJArWx. You'll see that you can download either the runtime or the SDK (which includes the runtime). Select the SDK downloads appropriate for your machine.

Be sure to install the UWP tooling for Visual Studio, which you can do by selecting the UWP workload in the Visual Studio installer. If you don't have the Windows 10 SDK build 16299 enabled in the workload, please make sure to do that.

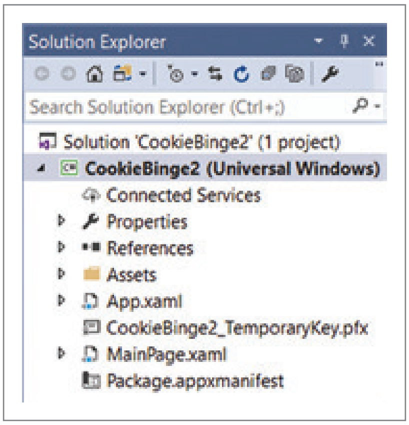

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'CookieBinge2' (1 project)
CookieBinge2 (Universal Windows)
Connected Services
Properties
References
Assets
App.xaml
CookieBinge2_TemporaryKey.pfx
MainPage.xaml
Package.appxmanifest

Figure 2 **The Project as Initially Created by the Template**

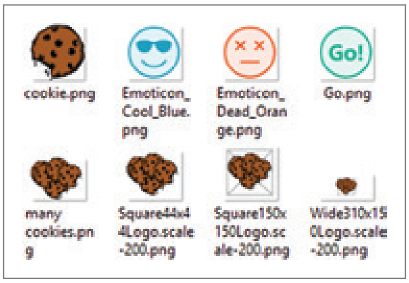

cookie.png
Emoticon_Cool_Blue.png
Emoticon_Dead_Orange.png
Go!
Go.png
manycookies.png
Square44x44Logo.scale-200.png
Square150x150Logo.scale-200.png
Wide310x150Logo.scale-200.png

Figure 3 **The Images You'll Be Copying into the Assets Folder**

An Early-Adopter Caveat

Note that as I'm writing this, EF Core 2.0 is not wholly aligned with the new version of UWP, though some fixes are coming soon in the 2.0.1 patch. For example, there's a current bug with executing relational queries when in a UWP project. My sample avoids this problem. You can track the issues on the EF Core GitHub repository at bit.ly/2xS35Mq.

Creating the New UWP Project

Once all of the proper tooling is installed, Visual Studio will display a set of UWP project templates in a section named Windows Universal. From that set of templates, I'll create a new Blank App (Universal Windows). The framework dropdown defaults to 4.7.1 (although the .1 is cut off in **Figure 1**) and I'll leave that default as the base version of .NET Framework for my app. I named the new project CookieBinge2.

You'll be prompted to select a target version and a minimum version of UWP. Be sure to choose Windows 10 Fall Creators Update for both. The current build number is 16299.

When the project is created, you'll see the files and folder in Solution Explorer, as shown in **Figure 2**.

Because this column is focused on data, not UI building, I'll take some shortcuts to get the UI in place, then add in the APIs and code for EF Core and its data-persistence activities.

You'll find the necessary code to copy and paste in the download associated with this column. To start, delete the images in the new project's Assets folder and in their place, copy the images from the Assets folder (shown in **Figure 3**) provided in the download.

Adding in Persistence Logic with EF Core 2.0

You'll start by adding two classes to the project, as follows.

The CookieBinge.cs class is not much more than a DTO, containing only properties that need to be stored. There's no real logic that needs to be performed. Here's the class:

```
public class CookieBinge {
  public Guid Id { get; set; }
  public DateTime TimeOccurred { get; set; }
  public int HowMany { get; set; }
  public bool WorthIt { get; set; }
}
```

The BingeService class encapsulates the tasks I determined this game needs to perform, serving to marshal information between the UI and the data store. BingeService has three methods: RecordBinge (to persist the results locally), GetRecentBinges (to retrieve a subset of the game scores) and ClearHistory (to clear all results from the local store). Here's the class before the logic of those methods has been implemented:

```
public static class BingeService {
  public static void RecordBinge(
    int count, bool worthIt) { }
  public static IEnumerable<CookieBinge>
    GetRecentBinges(int numberToRetrieve)
  public static void ClearHistory() {  }
}
```

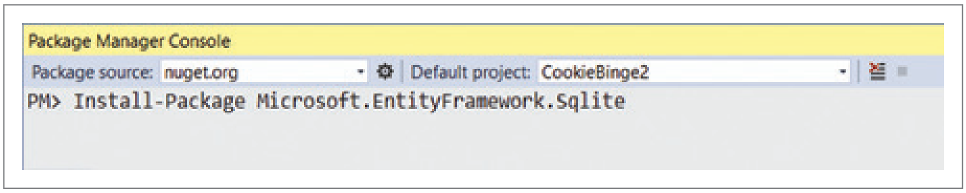
In order to flesh out the service, I need to add in the persistence functionality. This is where EF Core 2.0 comes in to read and store the binge results to the device. EF Core has a provider for SQLite databases, which can be used directly on a variety of devices, so that's what I'll use for this demo.

Now that I've chosen the data store, I can add the EF Core provider for SQLite into my app and it will, in turn, pull in the related EF Core packages it relies on, including SQLite if needed. EF Core is no longer one big package. Instead, the logic is divided into various assemblies, so you can minimize its footprint on your device (or server or wherever you deploy it) with only the subset of logic your application requires.

In Visual Studio 2017, you use the NuGet Package Manager to add NuGet packages into your projects—either visually or via PowerShell commands such as install-package.

In Visual Studio 2017, you use the NuGet Package Manager to add NuGet packages into your projects—either visually or via PowerShell commands such as install-package. I'll use the PowerShell commands, which means opening the Package Manager Console (Tools | Manage NuGet Packages | Package Manager Console). As I have only one project in my solution, the console should indicate that CookieBinge2 is the default project, as shown in **Figure 4**. There I can install the SQLite provider package.

When the installation is complete, you'll see this package listed in the project references in Solution Explorer. For the curious,



```
PM> Install-Package Microsoft.EntityFramework.Sqlite
```

Figure 4 **Installing the SQLite Package in the Package Manager Console Window**

if you force Visual Studio to show all files, expand the obj folder and then open the project.assets.json file, you'll be able to see the dependencies that were pulled in by the SQLite provider, such as Microsoft.EntityFrameworkCore, Microsoft.EntityFrameworkCore.Relational and others. With EF Core and the SQLite provider in place, you can now create the DbContext. Add a new file named BingeContext.cs and copy the code listed in **Figure 5**.

Because this DbContext is in the UWP project, I'm taking advantage of available resources such as System.IO and Windows.Storage APIs. The download for this article will reflect this pattern. In contrast, another version of the app on my GitHub account (bit.ly/2liqFw5) has the back-end logic and EF Core in a separate .NET Standard Class Library project and uses dependency injection to provide the file path to the context. If you plan to use EF Core migrations to evolve the model and database schema, you'll need to use that separated architecture.

The context has a single DbSet to interact with CookieBinge data. The code in the OnConfiguring method lets EF Core know that it will use the SQLite provider and specifies a local file path for storing the data on the device where the app is running.

Now that the context is there, I can flesh out the BingeService methods. RecordBinge will take in values sent from the UI, build up a CookieBinge object and pass that on to the BingeContext to store into the local database:

```
public static void RecordBinge(int count, bool worthIt) {
  var binge = new CookieBinge {
                HowMany = count,
                WorthIt = worthIt,
                TimeOccurred = DateTime.Now};
  using (var context = new BingeContext()) {
    context.Binges.Add(binge);
    context.SaveChanges();
  }
}
```

The GetRecentBinges method will retrieve data from the locally stored database using the BingeContext and pass it back to the UI that made the request and has a panel to display that data:

```
public static IEnumerable<CookieBinge> GetRecentBinges(
  int numberToRetrieve) {
  using (var context = new BingeContext()) {
    return context.Binges
              .OrderByDescending(b => b.TimeOccurred)
              .Take(numberToRetrieve).ToList();
  }
}
```

The final method in BingeService, ClearHistory, empties out the local database completely—in case you don't want to be tormented by the excessive amount of binging you've done:

```
public static void ClearHistory() {
  using (var context = new BingeContext(){
    context.Database.EnsureDeleted();
    context.Database.EnsureCreated();
  }
}
```

This takes care of the back end of the game's logic.

The UI and Its Logic

The front end consists of the UI (MainPage.xaml), UI logic (MainPage.xaml.cs) and the BingeViewModel.cs file. There are a number of methods in these classes of interest to the functionality I've created in the back end. In MainPage.xaml.cs, there are two methods that call into the BingeService class. The ReloadHistory method calls the GetRecentBinges method and binds the results to a list on the UI:

```
private void ReloadHistory() {
  BingeList.ItemsSource = BingeService.GetRecentBinges(5);
}
```

This separation of concerns that results in passing data from one file to another allows for future maintenance of this app to be simpler.

The ClearHistory method calls the service method to purge all of the data from the local database and then forces the UI to reload its display of the history, which will now be empty:

```
private void ClearHistory_Click(object sender, RoutedEventArgs e) {
  BingeService.ClearHistory();
  ReloadHistory();
}
```

There are also methods in the UI that respond to the user clicking the Worth It and Not Worth It buttons to store the results, but these don't call the service directly. Instead, they call into the BingeViewModel method, StoreBinge, which then calls the service method to store the data. The StoreBinge method first calls the service method, passing along the data from the UI:

```
BingeService.RecordBinge(_clickCount, worthIt);
```

Figure 5 **The DbContext Class to Manage Persistence with EF Core**

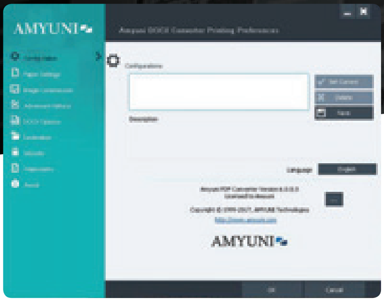
```
using System;
using System.IO;
using Microsoft.EntityFrameworkCore;
using Windows.Storage;

namespace CookieBinge2 {
  public class BingeContext:DbContext {
    public DbSet<CookieBinge> Binges { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder options){
      var dbPath = "CookieBinge.db";
      try {
        dbPath = Path.Combine(ApplicationData.Current.LocalFolder.Path, dbPath);
      }
      catch (InvalidOperationException){
        #TODO handle this exception
      }
      options.UseSqlite($"Data source={dbPath}");
    }
  }
}
```

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

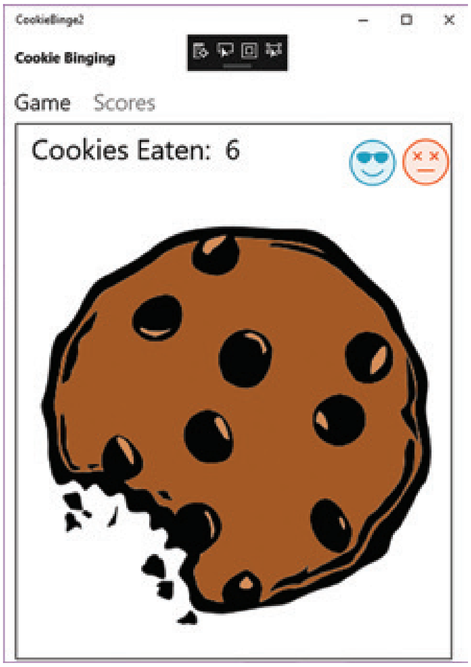
Configure the way the fonts are embedded into the DOCX file for optimal formatting

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

Powered by Amyuni Technologies:

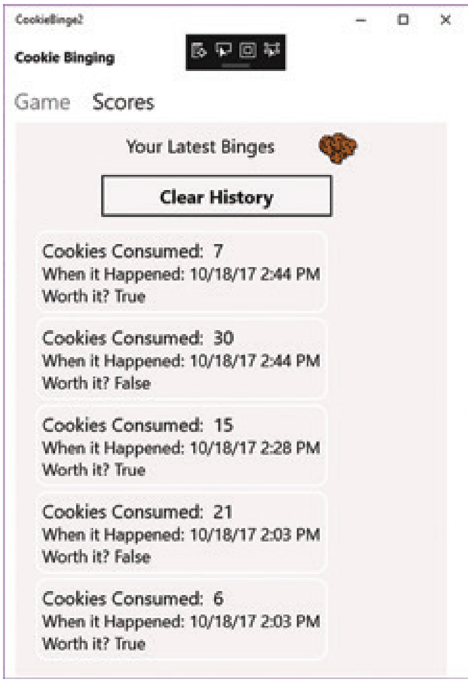
Developers of the Amyuni PDF Converter and Amyuni PDF Creator products integrated into hundreds of applications and installed on millions of desktops and servers worldwide.

www.docxconverter.com



Cookies Eaten: 6

Figure 6 **Binging on the Cookie; Six Eaten So Far**



Your Latest Binges

Cookies Consumed: 7
When it Happened: 10/18/17 2:44 PM
Worth it? True

Cookies Consumed: 30
When it Happened: 10/18/17 2:44 PM
Worth it? False

Cookies Consumed: 15
When it Happened: 10/18/17 2:28 PM
Worth it? True

Cookies Consumed: 21
When it Happened: 10/18/17 2:03 PM
Worth it? False

Cookies Consumed: 6
When it Happened: 10/18/17 2:03 PM
Worth it? True

Figure 7 **The History Stored in the Local Database via EF Core**

StoreBinge also performs some additional magic in the UI to clean up the just-finished binge and prepare the UI for a new binge. You can see this additional code in the download.

This separation of concerns that results in passing data from one file to another allows for future maintenance of this app to be simpler. Fortunately for me, the code changes as I've transitioned the app from EF7 to EF Core 1 to EF Core 2 have been minimal.

> In Visual Studio, you target either the local machine or the simulator, which opens up a separate window that simulates your local machine, to run or debug the app.

There's one last task to perform, which is to trigger the application to make sure that the local database file exists. In the App.xaml file, add the following code into the constructor method after this.Suspending += OnSuspending:

```
using (var context = new BingeContext()) {
  context.Database.EnsureCreated();
```

In the separated architecture on GitHub where you can use migrations, you'll see a call to the Migrate command after EnsureCreated to make sure any model changes are applied.

Running the App

In Visual Studio, you target either the local machine, or the simulator, which opens up a separate window that simulates your local machine, to run or debug the app. There's also an emulator for HoloLens available at bit.ly/2p8yRMf. I'll just run my app on the local machine. **Figure 6** shows the game play with a hint of the computer desktop background around the edges. Each click on the cookie displays the word "Nom!" at the mouse pointer and increments the "Cookies Eaten" counter. When satiated, the user clicks either the cool blue icon or the dead as a doornail icon to complete the game. **Figure 7** shows the Scores page, displaying the last five scores along with the Clear History button. This UI is just what a data geek could hack together. For proper UWP UI design guidance, you'll find a great place to get started at bit.ly/2gMgE74.

Coming Up: Sharing with Other Players via Azure Functions and Cosmos DB

If you can forgive the simplicity of my little game and design, the focus of the lesson here is that EF Core can now work directly on a device because it now relies on .NET Standard, not the full .NET Framework, and UWP supports .NET Standard. This means you can now use EF Core 2 in UWP apps on a variety of Windows 10 devices, such as HoloLens, Xbox, Windows Mixed Reality and more.

I've used EF Core 2.0 to store data locally on the device where the CookieBinge game is being played. In my next column I'll prepare for a new set of capabilities to the game by tying it to the Web. Using Azure Functions, the app will be able to send scores to an Azure Cosmos DB database, allowing users to compare their own scores across the various UWP devices they play on, as well as compare their cookie binging to other bingers around the world. Azure Functions and Azure Cosmos DB will keep me covered as my Cookie Binge game dominates the globe and needs to scale at the click of a button. ■

JULIE LERMAN *is a Microsoft Regional Director, Microsoft MVP, software team mentor 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 datafarm.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: *Clint Rutkas*

THE WORKING PROGRAMMER

TED NEWARD

How To Be MEAN: Angular Forms, Too

Welcome back again, MEANers.

In the last column (msdn.com/magazine/tktktktk), I talked about how to create forms with Angular, but candor compels me to admit a rather severe fact: I've really only begun to scratch the surface of the capabilities of Angular regarding forms, input, validation and more. For example, in the previous version of Angular (the not-at-all-confusing version called AngularJS), it was possible to set up "two-way binding" so that form input modified an ECMAScript object held in memory within the browser, which meant code could worry about model objects and not the input fields containing the data.

Angular hasn't abandoned that concept, but it was necessary to show you the syntax for event binding against input controls before we could get there. Now that you've seen it, you can begin to explore Angular form capabilities in greater depth.

SpeakerUI

In the last column, I showed you how to build a component called SpeakerEdit that captured user input. Bah. How entirely un-component-oriented of me. What you really should want, if this is going to take a truly component-oriented approach, is to encapsulate the details of displaying or editing a Speaker. That is, you want a component that can be handed a Speaker model object (for edit/update or even delete), a Speaker ID (meaning the Speaker model object needs to be fetched from the server), or "undefined" (in which case you're asking the user to populate a new Speaker model object), and "do the right thing" in each case. In short, the component should know already what to display and how to display it, based solely on the contents (or lack thereof) of a corresponding Speaker object.

Let's start by generating a SpeakerUI component. (I tend to use "UI" as a suffix, to indicate that this is intended as a standalone UI component around the Speaker model type. Some readers may prefer "SpeakerDetail," because this is allowing for the view or edit of the different details of a Speaker model. As always, whichever convention you choose, stick with it.) This is comfortable territory for you by now: "ng generate component SpeakerUI."

However, I'm going to make a slight addition to Speaker, to help show off some of the power of the TypeScript language when it comes to UI. To demonstrate a few different things, I'm going to mention that in between the last column and this, the application's client has said that Speakers have one or more Subjects on which they speak, like so:

```
import { Upvote } from './upvote/upvote';
import { Subject } from './subject';

export class Speaker {
  id: number;
  firstName: string;
  lastName: string;
  votes: Upvote;
  subjects: /*something*/[];
}
```

The Subject type will be defined in subject.ts, but you have a couple of options here. If subject makes sense to model as a string—meaning the actual value will be a raw, unadorned TypeScript string—but can only be of a few values, then you can use a TypeScript "string literal type" in TypeScript, like so:

```
export type Languages = "C#" | "Visual Basic" |
  "F#" | "ECMAScript" | "TypeScript"
```

Essentially, it's an enumerated string—instances of Subject can only be populated with the values specified in the "or"-style assignment expression, in this case, one of five different Microsoft languages. That said, however, as of TypeScript 2.4, it's also possible to write a full-blown enumerated type using string backing for storage, like so:

```
export enum Subject {
  CSHARP = 'C#', FSHARP = 'F#',
  VB = 'Visual Basic', ES = 'ECMAScript',
  TS = 'TypeScript'
}
```

This will have a few benefits when you get to modeling the checkboxes that will hold these five values for selection; the drawback to using the enumerated type, of course, is that it cannot hold anything other than one of the five possible Subject values. (Languages is, at heart, a string, and so any one of its five values can be stored as a string, which could of course be in a dropdown, or you could have an open-ended edit field allow users to type in other values beyond these.)

Let's assume the enum is the preferred choice for now, and that the Speaker has an array of Subject instances, called "subjects." That's all to be done with that for now.

Editing? Or Just Viewing?

From a UI perspective, it's usually preferential if the UI component can know whether the instance being displayed is being edited or just examined; sometimes you'll merely want to display the details of the Speaker, whereas at other times, you'll want to be able to edit them. The UI will often want to display the details differently when being edited (using edit field controls) than when being viewed (just leaving it as plain text) so as to provide a clear hint to the user when something is editable. Angular supports this, in a slightly roundabout way.

To begin, you need the component to know whether it's readonly or not; the easiest way to model that is to have a private Boolean field in the component by that exact name:

```
@Component({
  selector: 'app-speaker-ui',
  templateUrl: './speaker-ui.component.html',
  styleUrls: ['./speaker-ui.component.css']
})
export class SpeakerUIComponent implements OnInit {

  @Input() model: Speaker | number | undefined;
  public readonly = true;
```

Defaulting readonly to true is a matter of personal taste or application context.

Once that field is established, the template can differentiate between the two states, and choose between one of two different div sections in the template by setting the div hidden attribute to true or false accordingly:

```
<form>
  <div [hidden]="readonly">
    ...
  </div>
  <div [hidden]="!readonly">
    ...
  </div>
</form>
```

Here, the first section is hidden if the readonly flag is set; that makes it the editable section displayed; the second section, therefore, is the read-only section (that's to say, it's not hidden if the component isn't read-only). Notice how the property-binding syntax (the square brackets) is used to bind to the component's readonly field, inverting it when necessary using the Angular expression syntax.

Viewing

The second readonly section is pretty straightforward: Simply use the Angular double-bracket syntax to display the values of the model's fields, and display a button labeled Edit to kick the component over into editing mode when the user wants to edit the Speaker instance:

```
<form>
  <div [hidden]="readonly">
    ...
  </div>
  <div [hidden]="!readonly">
    FirstName: {{model.firstName}}<br>
    LastName: {{model.lastName}}<br>
    Subjects: <span>{{model.subjects}}</span><br>
    <button (click)="edit()">Edit</button>
  </div>
</form>
```

Figure 1 **Using the ngModel Attribute**

```
<form #speakerForm="ngForm">
  <div [hidden]="readonly">
    FirstName: <input name="firstName" type="text"
                 [(ngModel)]="model.firstName"><br>
    LastName:  <input name="lastName" type="text"
                 [(ngModel)]="model.lastName"><br>
    Subjects: <span>{{model.subjects}}</span><br>
    <button (click)="save()"
            [disabled]="!speakerForm.form.valid">Save</button>
    <button [disabled]="speakerForm.form.pristine"
            (click)="cancel()">Cancel</button>
    <br><span>{{diagnostic}}</span>
  </div>
  <div [hidden]="!readonly">
    . . .
  </div>
</form>
```

The edit function on the component will flip the readonly field to false, but this brings up an interesting question of functionality. Normally, when some kind of editing is presented to the user, there's also an opportunity to undo that editing and go back to the original state via a Cancel button. If that functionality is wanted or needed here, then you need to cache off the original values for later retrieval if the user selects Cancel. Therefore, a new field, cached (of Speaker type), should be added and the current model values copied over into it:

```
edit() {
  this.readonly = false;

  this.cached = new Speaker();
  this.cached.id = (<Speaker>this.model).id;
  this.cached.firstName = (<Speaker>this.model).firstName;
  this.cached.lastName = (<Speaker>this.model).lastName;
  this.cached.votes = (<Speaker>this.model).votes;
  this.cached.subjects = (<Speaker>this.model).subjects;
}
```

This will bring the UI into editing mode.

Editing

Editing, then, uses input fields instead of the double-bracketed syntax, but Angular holds another surprise: You can use the ngModel directive to help Angular provide some additional form-relevant behavior, such as automatically double-binding the model (the Speaker object) to the various form fields, and giving some predefined behavior for when fields get edited.

Defaulting readonly to true is a matter of personal taste or application context.

The first step is to tell Angular that this is a form for which model support is wanted; this is done by creating a template variable reference for the form object on the form tag in the template:

```
<form #speakerForm="ngForm">
  ...
</form>
```

This tells Angular that you want Angular to "do its form thing." From there, you can use the ngModel attribute on the different input fields to provide clear binding from field to model without requiring any additional work, as shown in **Figure 1**.

There's several interesting things going on in Figure 1.

First, notice the "{{diagnostic}} at the bottom of the form; this is a useful trick to see what's going on inside the component as you're editing/viewing the component. To do this on your own components, define a property-get method called diagnostic that dumps interesting elements:

```
get diagnostic() {
  return 'readonly:' + this.readonly + ';'
    + 'cached:' + JSON.stringify(this.cached) + ';'
    + 'model:' + JSON.stringify(this.model);
}
```

This is useful to see that Angular literally changes the model object in memory in response to each and every keystroke that appears in the input forms. (It's also useful to help make sure that the caching behavior—for cancellation support—is behaving correctly, though realistically that should be unit-tested before accepting it as genuine and bug-free.)

Second, notice how the firstName and lastName input fields have a dual-mode binding attribute on them called ngModel, referencing the property of the model object to which these fields should be bound. The dual-mode binding (the [(…)] syntax) suggests that changes to either the underlying model or the UI will be automatically reflected in the other, leaving little to do programmatically here. This is almost identical to the two-way binding from AngularJS, which was always one of the strengths of that framework.

Finally, notice how the Save button is only enabled if speakerForm.form (the form object defined via the # syntax in the form element) has a valid property that's false. Valid is one of several properties the form exposes to indicate whether the form would require a save. Several other properties are present, including the pristine property referenced in the Cancel button, but that discussion has to be deferred until the next column, when I talk about form validation.

The save and cancel methods, then, are pretty easy to imagine:

```
save() {
  this.cached = undefined;
  this.readonly = true;
}
cancel() {
  this.readonly = true;

  // Bring back cached values
  this.model = this.cached;

  // Clear out cache
  this.cached = undefined;
}
```

Again, all of this is simply manipulating the two-state state machine to go back and forth between editing and viewing mode with cancel support.

Wrapping Up

With a quick dip of the pen (so to speak), I've opened up a fairly large subject, to be sure. Angular's support for form-based input is extensive, to say the least, particularly when combined with some of the component lifecycle methods, such as ngOnInit, where some initialization will usually take place to establish the initial contents of the UI. In the SpeakerUI, for example, ngOnInit will look at the model input property, and if it's undefined, assume a new Speaker is being created; if it's a number, assume that ID from the SpeakerService needs to be fetched; and if it's a Speaker object, assume that's the object wanted to use as the model. It would be reasonable to assume that other lifecycle methods could play a part in the component's interaction with the surrounding system, as well, depending on circumstances.

In the next column, I'll look at the form validation capabilities of Angular, so that both a first and a last name must be required of the speakers, among other things. I'll also examine how new subjects on Speakers can be captured, and toss in the Upvote component that was constructed oh-so-long ago. In the meantime, however … Happy coding!

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor, currently working as the director of Developer Relations at Smartsheet.com. He has written a ton of articles, authored and co-authored a dozen books, and speaks all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.*

Thanks to the following technical expert for reviewing this article:
Garvice Eakins

ARTIFICIALLY INTELLIGENT

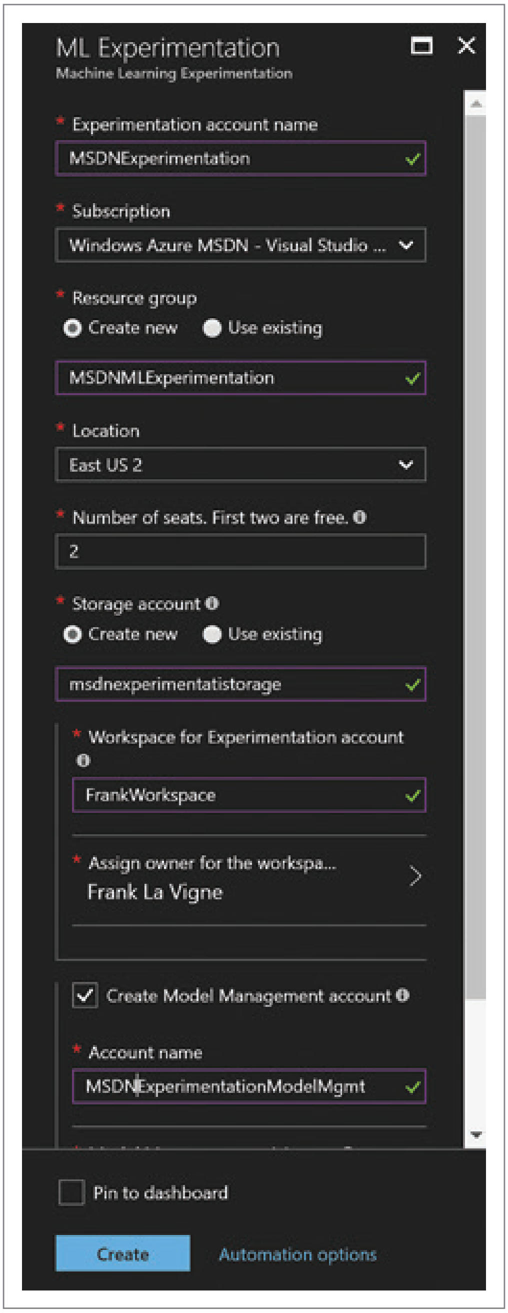
FRANK LA VIGNE

Exploring the Azure Machine Learning Workbench

In the last two columns, I explored the features and services provided by Azure Machine Learning Studio. In September 2017, Microsoft announced a new suite of tools for doing machine learning (ML) on Azure. The cornerstone of these new tools is Azure Machine Learning Workbench. However, what could be better for doing ML than the simple drag-and-drop interface of Machine Learning Studio?

> In September 2017, Microsoft announced a new suite of tools for doing machine learning on Azure. The cornerstone of these new tools is Azure Machine Learning Workbench.

Machine Learning Studio is an ideal tool for creating ML models without having to write code, but it falls short in several areas. First and foremost, the tool's simplicity requires a "black box" approach. There's no visibility into the algorithms being used, and only parameters that are exposed in the UI can be manipulated. Source control is effectively nonexistent. While the Run History makes it possible to access previous versions of a project, Machine Learning Studio lacks integration with conventional source control tools such as Git. Finally, managing the deployment of models exposed as Web services poses unique challenges at scale.



ML Experimentation
Machine Learning Experimentation

* Experimentation account name: MSDNExperimentation

* Subscription: Windows Azure MSDN - Visual Studio ...

* Resource group: ● Create new ○ Use existing
MSDNMLExperimentation

* Location: East US 2

* Number of seats. First two are free.: 2

* Storage account: ● Create new ○ Use existing
msdnexperimentatistorage

* Workspace for Experimentation account: FrankWorkspace

* Assign owner for the workspa...: Frank La Vigne

☑ Create Model Management account

* Account name: MSDNExperimentationModelMgmt

☐ Pin to dashboard

Figure 1 **Options Selected to Create a Machine Learning Experimentation Service**

Since the Machine Learning Studio launch in 2015, the team has been gathering feedback from users large and small. They've taken that feedback and created something quite unique with Machine Learning Workbench—a tool that satisfies the needs of ML professionals of all levels.

Create Machine Learning Accounts

Machine Learning Workbench requires the provisioning of Machine Learning accounts in Azure. Go to the Azure Portal (`portal.azure.com`) and click the New button in the upper-left corner. In the textbox that appears type "Machine Learning Experimentation" and click on the first result. On the following blade, click on the Create button at the bottom of the screen.

Next, a blade appears that asks for some information used to create the service. You may choose any name that meets the validation requirements. Please refer to **Figure 1** to see the values I chose. Currently, this service is only available in three geographies. As I'm located on the East Coast of the United States, I selected East US2. Ensure that the checkbox next to Create model management account is checked. Next, choose a pricing tier. For the purposes of this article, the free DevTest tier will suffice. Click on

ComponentSource® /update/2017/12

Experience the brand new look at **www.componentsource.com**

DevExpress DXperience 17.1 | from $1,439.99

DevExpress

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

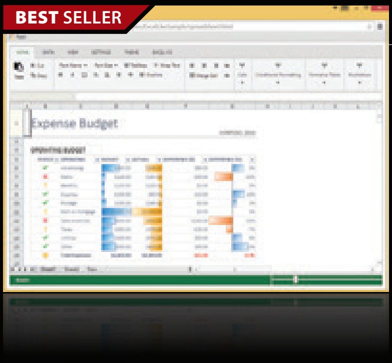
- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context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

LEADTOOLS Medical Imaging SDKs V19 | from $4,995.00 SRP

LEADTOOLS THE WORLD LEADER IN IMAGING SDKs

Powerful DICOM, PACS, and HL7 functionality.

- Load, save, edit, annotate & display DICOM Data Sets with support for the latest specifications
- High-level PACS Client and Server components and frameworks
- OEM-ready HTML5 Zero-footprint Viewer and DICOM Storage Server apps with source code
- Medical-specific image processing functions for enhancing 16-bit grayscale images
- Native libraries for .NET, C/C++, HTML5, JavaScript, WinRT, iOS, OS X, Android, Linux, & more



SpreadJS | from $984.02

GrapeCity.

Deliver intuitive, efficient, multi-functional, pure JavaScript spreadsheets for Enterprise apps.

- Harness the power of a spreadsheet to display and manage data like Microsoft Excel
- Go beyond the grid with cards, trellis, calendar, Gantt, news feed, timeline and more
- Renders to the HTML canvas for a fast, interactive user experience across all browsers
- Modularized - so you only need to load the JavaScript that contains the features you need
- A Client-side component - works with Windows, Linux, MacOS, Android and iOS

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

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

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

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

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

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

VISA DISCOVER

GSA Schedule
Contract GS-35F-0188R

the DevTest tier and click Select. Last, click on the Create Button.

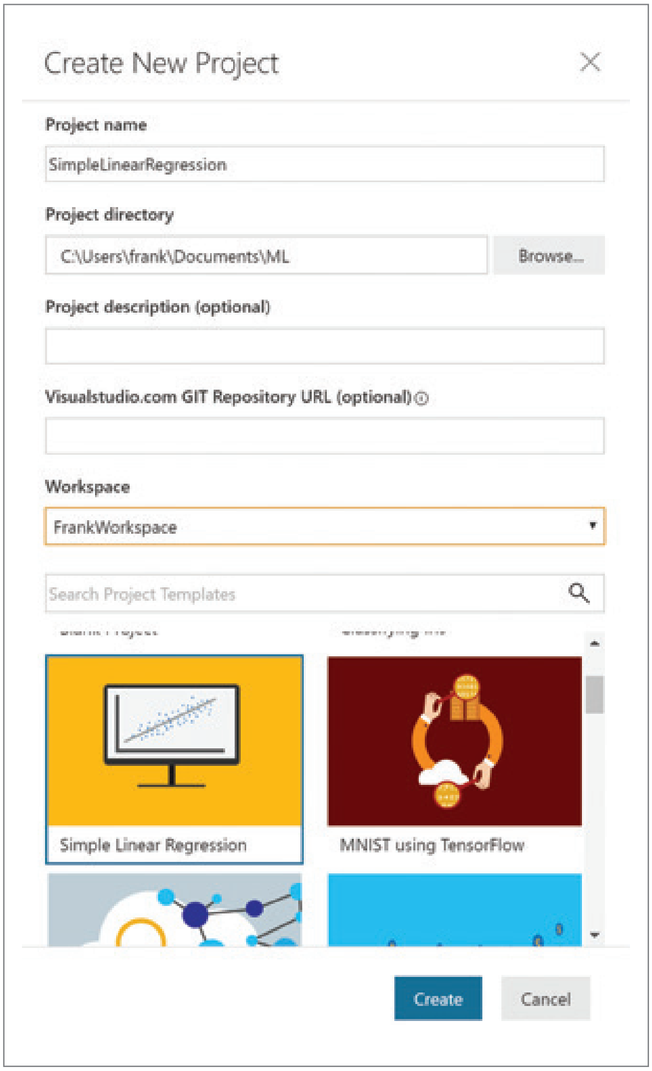
While the cloud services initialize, it's a good time to set up your local computer.

Installing Machine Learning Workbench

While Machine Learning Workbench is available on Mac and Windows, this article follows the path for a Windows install. For further details and system requirement to install on a Mac, please refer to Microsoft's documentation on the topic at bit.ly/2zYrA7X.

If you haven't already installed Docker on your system, now would be a good time to do so. While this article will not make use of it, future articles on Machine Learning Workbench will. Machine Learning Workbench uses Docker to run code locally in different frameworks. Some projects have Python and PySpark containers available by default, while others just have Python containers. Available container configurations can be managed by configuration files.

Machine Learning Workbench for Windows can be downloaded from aka.ms/azureml-wb-msi. Once the installer downloads, run it and follow the directions to install the tool. When the install is complete, launch the program. Enter the same account credentials used to create the Machine Learning Experimentation Service in Azure. After a moment, the screen updates to show the Get Started window. Note in the Projects area to the left that the name of the workspace should match the value entered previously into Azure. Also take note that in the bottom left-hand corner of the screen, a teal button appears with the initials of the active account. Click on that button to reveal a fly-out menu where you can see the current account and sign out.



Create New Project

Project name: SimpleLinearRegression

Project directory: C:\Users\frank\Documents\ML

Project description (optional)

Visualstudio.com GIT Repository URL (optional)

Workspace: FrankWorkspace

Figure 2 **Create New Project Dialog**

Creating and Running a Project

To create a new project, either choose New Project from the File menu or click the plus sign in the Projects pane. In the following dialog box, shown in **Figure 2**, enter SimpleLinearRegression as the project name, choose a directory for the project files, and select Simple Linear Regression from the available project templates. Project templates are a great resource for learning how to create an ML project based on various types of models and external libraries. Click the Create button to create the project.

Once the project is created, the project dashboard opens. Each project in Machine Learning Workbench opens to its project dashboard. The dashboard contains instructions and information about the project and functions very much like a readme.md file in a GitHub repository. In fact, the contents of the dashboard file are located in the readme.md file in the root directory of the project.

Choose Open Command Prompt from the File menu. In the command prompt type:

```
conda install matplotlib
```

Follow the instructions on screen if prompted to install matplotlib, a graphing library for Python. Once the install completes, close the command prompt window to return to the Machine Learning Workbench. Take note of the Run button and the controls to its left. Make sure that local and linar_reg.py are selected in the two dropdowns and the Arguments text box is empty. Click Run.

Immediately after clicking Run, a Jobs pane appears with one entry in the list. After a few moments, the job will complete.

Project templates are a great resource for learning how to create an ML project based on various types of models and external libraries.

Exploring the Interface - Jobs

To the left of the screen, there's a vertical toolbar, click on the fourth icon from the top, which resembles a clock with a counterclockwise arrow around it. In this case, the linear_reg.py file failed to run twice before executing successfully, as seen in **Figure 3**. Note that Machine Learning Workbench tracks the jobs executed through it. At the bottom of the window under the STATUS header, click the word Completed to view the properties of the run.

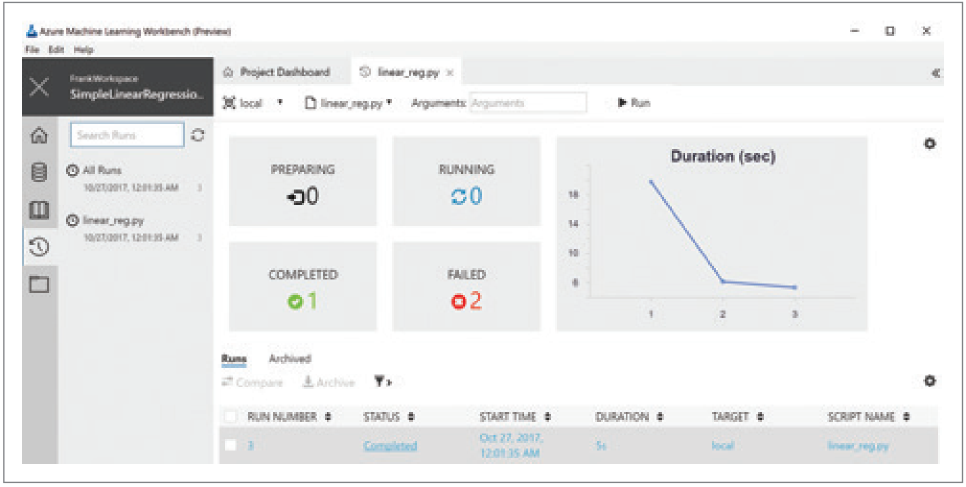
Take some time to explore the various portions of this screen: Run properties, Outputs, Visualization and Logs. Run properties displays data about the properties of this particular run. If the Python script creates output files and places them in an outputs subfolder, they'll appear in the list under the Outputs section. In this run, there's one file: lin.png.

Further down the screen is the Visualization section, which displays any images created by the Python script. For this script, there's one that plots the linear regression from the data files included in the project. This is the same file shown in the Outputs section: lin.png. Last, the Logs section lists all the logs associated with this run.

Exploring the Interface - Files

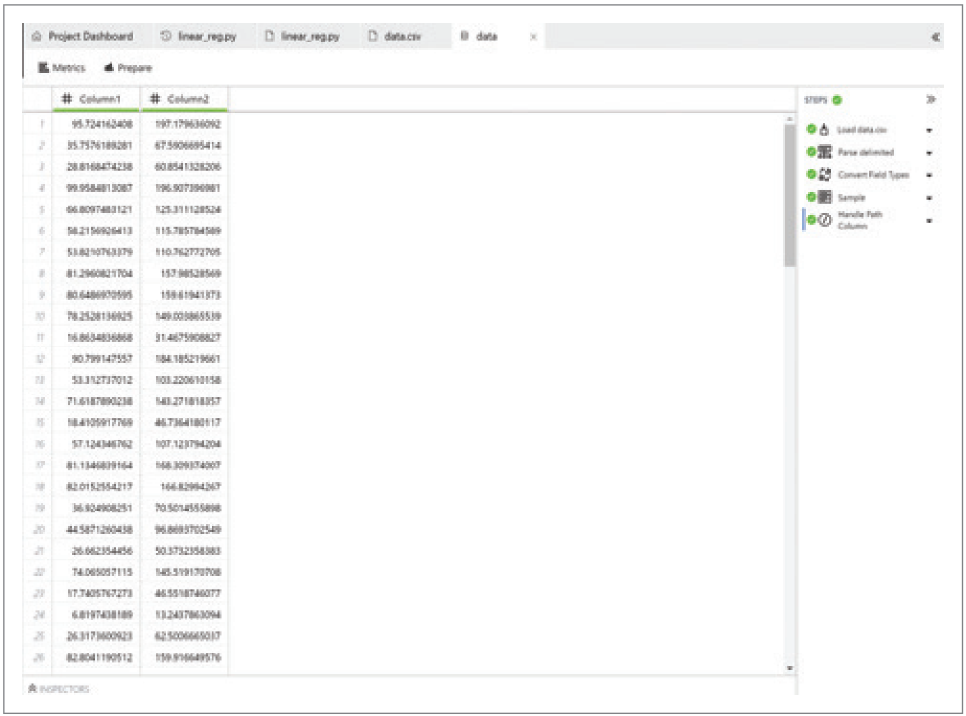
So far, I've run a Python script to perform a linear regression and produce an output visualization, but I haven't shown off any code. Let's take a look at the code behind the job that was just run. On the toolbar on the left-hand side of the screen click on the folder icon immediately below the Jobs icon. In the panel that appears, there's a list of files associated with this project: three directories and five files. Click on the linear_reg.py to view the contents of the script that was run earlier.

For readers unfamiliar with Python, the code first imports various libraries to do advanced mathematics with arrays (NumPy) and plot graphs (matplotlib). Next, the code loads the data from a local file, data.csv, into an array. Then the code converts the array into an nparray, which enables NumPy to perform matrix multiplication. After some calculations, the code computes a linear regression formula that approximates the data and prints out its findings. In lines 35 through 39, the code uses matplotlib to render a graph of the data and plot a line representing the linear regression. The rest of the code prints out information pertaining to the error and accuracy of the linear regression model.



RUN NUMBER	STATUS	START TIME	DURATION	TARGET	SCRIPT NAME
3	Completed	Oct 27, 2017, 12:01:35 AM	5s	local	linear_reg.py

Figure 3 **The Jobs Run History**



	Column1	Column2
1	95.724162408	197.179636092
2	35.7576189281	67.5906695414
3	28.8168474238	60.8541328206
4	99.9584813087	196.907396981
5	66.8097483121	125.311128524
6	58.2156926413	115.785784589
7	53.8210763379	110.762772705
8	81.2960821704	157.98528569
9	80.6486970595	159.61941873
10	78.2528136925	149.003865539
11	16.8634836868	31.4675908827
12	90.799147557	184.185219661
13	53.312737012	103.220610158
14	71.6187890238	143.271818357
15	18.4105917769	46.7364180117
16	57.124346762	107.123794204
17	81.1346839164	168.309374007
18	82.0152554217	166.82994267
19	36.924908251	70.5014555898
20	44.5871260438	96.8693702549
21	26.6623544456	50.3732358383
22	74.065057115	145.519170708
23	17.7405767273	46.5518746077
24	6.8197438189	13.2437863094
25	26.3173600923	62.5006665037
26	82.8041190512	159.916649576

Figure 4 **The Resulting Data Set**

Once you've had a chance to explore the script file, click on the data.csv file to view the data file. Note that the file immediately loads and displays a .csv file with two numbers per row separated by a comma.

Data Munging Tools Built In

The second icon from the top on the toolbar brings up the data pane. Clicking on it reveals a tree control with two empty nodes. The linear_reg.py file managed the data on its own and didn't make use of the Machine Learning Workbench advanced data munging tools. Now would be a good time to explore creating a data source through Machine Learning Workbench. Click on the plus sign in the data pane and click on Add New Data Source.

In the following dialog box, click on the box labelled File(s)/Directory and then click Next. Browse to the data.csv file in the project directory and click Open. Click Next once more. Note that Workbench has already detected that this is a comma-separated file. Leave all the settings as is and click Next. In this Data Types step, notice that Machine Learning Workbench automatically detected the formats of each data field. Click Next to view the Sampling step.

For this small data set, there's no need to change anything. For larger data sets, this screen allows you to craft a custom strategy to load data selectively. This way, when dealing with a multi-petabyte data set, you can selectively deal with a small portion of the data to define rules on how to transform the data. Click Next and leave the default selection to not include the path column in the final data set. Click Finish. The screen should look like **Figure 4**.

Note that the pane on the right side displays the steps just taken in a list. Clicking on any one of the items will reveal an option to edit the action. In more advanced scenarios, with more steps taken to munge the data, this feature allows for changing and editing the steps.

At the top of the screen, there are two toolbar buttons: Metrics and Prepare. Click on Metrics. In a few seconds, Machine Learning Workbench automatically generates a series of data visualizations for each column. When working with a new data set, it can be helpful to gather some basic information about each field. The metrics view creates histograms and basic descriptive statistics about every field in the data set. You can customize which metrics are displayed by clicking on the Choose Metric dropdown list. By default, all metrics are selected. Click on the newly visible Data toolbar button to return to the previous screen.

Now, click on the Prepare button. In the Prepare dialog, leave the top dropdown list at its default of New Data Preparation Package and enter SimpleDataPrep into the Data Preparation Package Name. Click OK. Now the Data pane has two entries

Right-click on the SimpleDataPrep entry and click Generate Data Access Code File. The newly created file opens, displaying stub code that loads a referenced package and returns a Pandas DataFrame. In a PySpark environment, this call returns a Spark DataFrame. DataFrames are a common data structure in Python. Using the DataFrame creation code generated by Machine Learning Workbench can save a great deal of time.

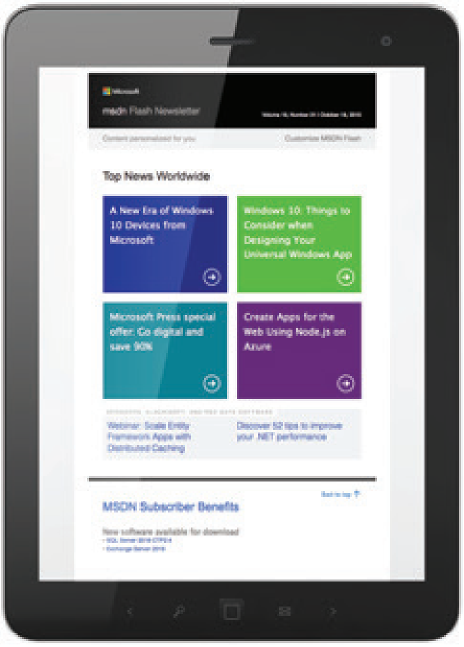
Wrapping Up

I was skeptical that Machine Learning Studio could be improved upon at first. However, the more I use Machine Learning Workbench, the more impressed I am with it. It not only provides mechanisms to import data, it also auto-generates a package that can clean data and expose it as a DataFrame, a common data format for Python. By importing data this way, data scientists can save a great deal of time. Furthermore, you saw that using the data import tool was not a difficult requirement. Machine Learning Workbench also allows for the use of any Python library, such as matplotlib.

This article barely scratches the surface of what Machine Learning Workbench is capable of. In future articles, I'll explore this great tool even more. For instance, configuring Machine Learning Workbench to work with virtual machines in Azure for fast processing, transforming data "by example," and working with Jupyter Notebookz inside Machine Learning Workbench. ■

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 experts for reviewing this article: *Andy Leonard (EnterpriseDNA), Hai Ning (Microsoft), Jonathan Wood (Wintellect)*



Modernize a .NET App with Docker and Windows Server Containers

Sean Iannuzzi

By now you've most likely heard of Docker, Docker containers and, with the introduction of Windows Server 2016, integrated Windows Server Containers. I truly believe that within a few years Docker containers will become the standard for how Web sites, applications and other systems run, as opposed to relying on running virtual machines (VMs) to support applications. Using Docker has enabled scalability, isolation, and security while also ensuring that applications and systems are configured properly with little support from a deployment standpoint. Compared to the complexity of setting up a VM and configuring all the required features, the simplicity of the Docker setup is very beneficial. Just as physical machine requests were gradually phased out in favor of VMs, it's more likely than not that Docker will begin to replace the need for VMs in the next few years—if not sooner.

This article discusses:

- Running a traditional .NET application in Docker
- Leveraging file sharing between the host system and a Docker container
- Enabling socket communication between the host and the Docker container

Technologies discussed:

Microsoft .NET Framework, Docker, Windows Containers, Windows Server 2016

Code download available at:

msdn.com/magazine/1217magcode

In this article I'll focus on how I leveraged a container approach, using Windows Server 2016, file sharing and socket communications with Windows Server Containers, to modernize several .NET applications. I'll provide details on how I used Windows PowerShell to create a Docker image and share files and sockets between a host Windows Server 2016 system and a Windows Server Container. It's likely that many of your applications have common functionality such as this that you'd need to enable to ensure your .NET application can be ported to a Docker container. Many of the features I review aren't Windows Server Container-specific and could be leveraged for any applications that have similar functionality.

The Business Challenge: A CPU-, Memory- and Disk-Intensive .NET App

My business challenge was to modernize several existing .NET and C++ console applications responsible for handling large volumes of data, which involved very heavy CPU-, memory- and disk-intensive processing. I needed to expose these console applications in a more traditional Web model where the system would migrate from a single-user system to a multi-user supported setup. Given how the applications were set up and the volume of data being processed, I didn't want to manage multiple copies of the data or executables across VMs.

As part of this business challenge, I needed to determine how I could best scale out these applications, as well as minimize network latency and file management across the network. Performance of the applications was critical, and any use of network sharing, file sharing or other distributed processing would significantly impact their performance. Therefore, in order for this business challenge to be considered successful, I needed to provide a scalable model that also yielded a high level of performance (with regard to CPU, memory and disk IO), without having to maintain multiple copies of my data. As with most projects, the timeline to deliver a newly modernized and scalable version of these applications was very limited, eliminating the possibility of a complete redesign.

Important Features: File Sharing, Socket Connections and .NET in Docker

For my particular applications, I considered several options prior to landing on using Docker and, more specifically, on Windows Server Containers. As part of my evaluation, I had three very specific technical challenges to prove out in order to migrate the applications successfully to Docker:

- Running a traditional .NET app in Docker.
- Leveraging file sharing between the host system and my Docker container.
- Enabling socket communication between the host and the Docker container.

I'll show you in detail how to overcome these technical challenges and how to implement the concepts with Docker and Windows Server Containers running on Windows Server 2016. The concepts themselves are just the beginning when considering how many of your .NET applications could potentially be migrated to Docker or Windows Server Containers. The examples I'll review can be applied or expanded more broadly to address various application features, which in turn can provide your applications with a more modernized deployment.

Application Performance: 8GB RAM, 10TB of File Processing

Before I dive too deeply into the options and concepts I considered, I want to provide a little more detail on the applications and systems that I moved to Docker containers. First and foremost, the applications are rather unique in the type of work they perform, and they're very process-, memory- and disk-intensive. Moreover, the speed at which the applications perform is critical to the success of the system.

My applications, primarily designed for a single user, perform very complex calculations processing data files and were built with a combination of C++ and the .NET Framework. To give you an idea of the performance challenges of my system, it takes approximately 8GB of RAM per user to perform calculations on data files that are upward of 10TB in size, and requires pre-allocated memory and extremely fast disk speeds to process the large volumes of data in seconds. The system also uses socket connections for invocation and notification from the requestor. As the applications and systems evolved, I found I needed a quick way to scale the system and support multi-user processing. I expect many of you can think of similar applications that might benefit by being moved into a container.

Solution Options: Reengineer, Auto-Scale, Docker

The technical challenges I faced involved evaluating the different ways I might achieve my goals. I considered three options.

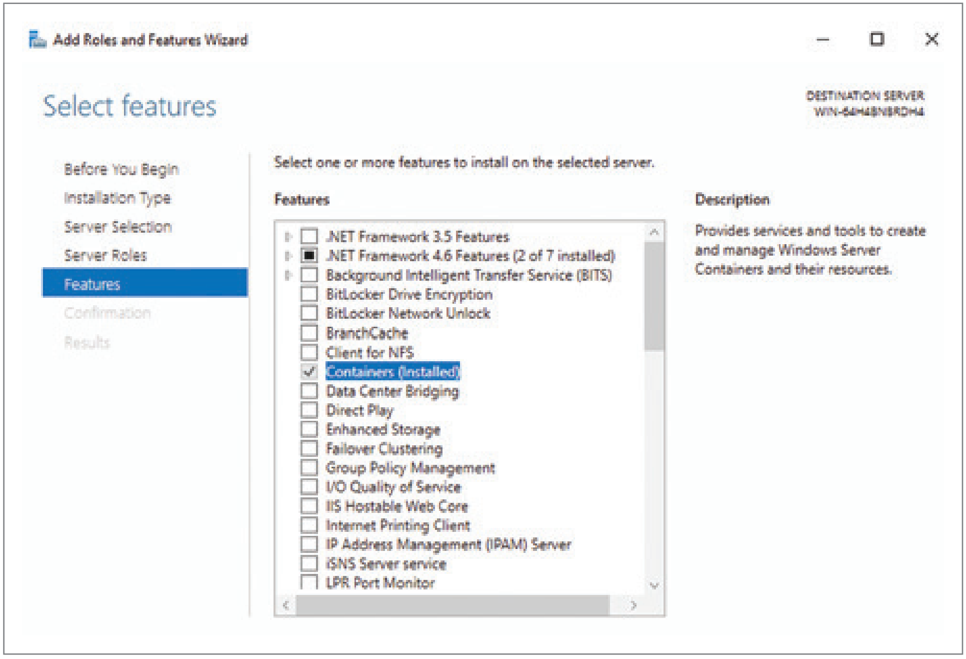
1. Reengineering: One option was to reengineer the entire application suite. This would surely work, but given the size and complexity of my system, I needed a solution that would introduce less risk and not take as long to complete. Waiting a year or even a several months to redesign the system was not going to be acceptable. However, it was still important to evaluate this option in the event it might turn out to be a reasonable solution.

2. Auto-Scaling: Another option was to evaluate how I could leverage VMs and auto-scaling. This would definitely be quicker than rewriting the application and would lessen the risk overall. However, it would add a lot of overhead because of the time it would take to allocate a VM, especially a VM with 10TB of storage. Even though I could find solutions for this, such as using standby instances and then handling the provisioning and de-provisioning of the servers via an additional layer or application, it still didn't seem like the best approach. However, this option was definitely moving me in the right direction because it didn't involve reengineering the entire application and could deploy multiple executables per VM and scale out the VMs automatically. I decided to continue my search for a simpler implementation model using a more modern technological approach.

3. Docker Container: The last option I considered was to use Docker, with interoperability between the host system and the Docker containers. Using Docker containers would allow me to scale the system as needed without having to reengineer the entire system. This approach would lessen the risks involved with reengineering the application, provide a level of isolation for security purposes, and allow me to implement these updates quickly while still providing the level of scale I needed.

Deploying .NET Apps with Docker

The main issues I had with the Docker option was that the application was written in .NET and C++ and I had concerns that my application wouldn't be able to be run in Docker directly. As soon as I began researching how to migrate my .NET/C++ apps to Docker, I learned that it would require an upgrade or redesign. Keeping in mind that my approach had to be quick, I began to learn more about Windows Server 2016 and the fully integrated Windows Server Containers. By leveraging Windows Server Containers, I was hoping I could leave the application as is and deploy all dependencies along with the other required setup in my container. The initial technical challenge I encountered was that traditional Docker containers for .NET apps require .NET Core, while my application was written with .NET and C++. Of course, I could've upgraded the application to .NET Core, but this would've involved a significant effort and I was trying to deploy a solution that was as quick as possible with the least amount of risk. I was also trying to ensure that I included the ability to scale, along with a level of isolation and security to my application, as well.



Select one or more features to install on the selected server.

Features

- ☐ .NET Framework 3.5 Features
- ■ .NET Framework 4.6 Features (2 of 7 installed)
- ☐ Background Intelligent Transfer Service (BITS)
- ☐ BitLocker Drive Encryption
- ☐ BitLocker Network Unlock
- ☐ BranchCache
- ☐ Client for NFS
- ☑ Containers (Installed)
- ☐ Data Center Bridging
- ☐ Direct Play
- ☐ Enhanced Storage
- ☐ Failover Clustering
- ☐ Group Policy Management
- ☐ I/O Quality of Service
- ☐ IIS Hostable Web Core
- ☐ Internet Printing Client
- ☐ IP Address Management (IPAM) Server
- ☐ iSNS Server service
- ☐ LPR Port Monitor

Description

Provides services and tools to create and manage Windows Server Containers and their resources.

Figure 1 **Enabling the .NET Framework and Container Services**

Although the use of Windows Server Containers was beginning to look very promising, I still needed to test a number of different concepts—such as file sharing and socket connections—that you might also find very useful. While much of what I describe is unique to my particular setup, the options and concepts are not and can be leveraged for other systems that need this type of migration or scale without having to redesign or rewrite the application. Of course, this approach doesn't replace an application redesign, but does offer time for a team to redesign the application if that's the desired direction. As part of that redesign, the team can reengineer the application that's Docker-compatible or -enabled.

In the next few sections I'll describe:

1. How I set up the Windows Server 2016 VM to support Windows Server Containers.
2. How I created my Docker image using PowerShell.
3. The Docker file based on Windows Server Core.
4. How to enable advanced file sharing between the host and the container.
5. How to enable a socket listener from the host and the container.

Figure 2 **PowerShell Functions to Build the Application and Create a Docker Image**

```
Set-StrictMode -Version Latest
$ErrorActionPreference="Stop"
$ProgressPreference="SilentlyContinue"
s
# Docker image name for the application
$ImageName="myconsoleapplication"

function Invoke-MSBuild ([string]$MSBuildPath, [string]$MSBuildParameters) {
  Invoke-Expression "$MSBuildPath $MSBuildParameters"
}

function Invoke-Docker-Build ([string]$ImageName, [string]$ImagePath,
  [string]$DockerBuildArgs = "") {
  echo "docker build -t $ImageName $ImagePath $DockerBuildArgs"
  Invoke-Expression "docker build -t $ImageName $ImagePath $DockerBuildArgs"
}
```

Windows Server 2016 and Containers

To get started I deployed a Windows Server 2016 VM and enabled the appropriate features, such as .NET Framework, IIS and containers, as shown in **Figure 1**.

Please note that in order to build this type of solution you must have the .NET Framework installed.

After installing all of the required features, I validated each of them accordingly. To make sure Docker was running properly I ran the PowerShell command docker –version. I then verified that the Windows Docker Engine Service was also running by typing "(get-service "Docker").Status" from PowerShell. As a final step, I performed a docker pull request for the Windows Server Core Docker image from `dockr.ly/2i7pDSn`. After the pull request completed I verified that the Docker image was created successfully by running the command docker images.

Once I installed the Windows Container Services and set up the environment with my base Docker image, I was ready to begin working with my .NET console application.

.NET App Setup

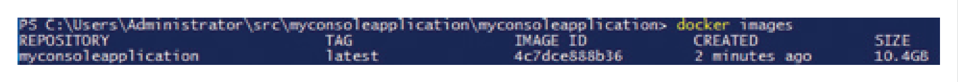
I started with a very basic console application using the .NET Framework 4.6.1. The application really didn't do much other than take an argument and display a response. Before going too far with the full migration to a Windows container, I wanted to make sure that the required functionality was going to work as intended. However, there were a number of steps I needed to take before I could run the application in a container on Windows Server 2016.

The first step was to create a reusable "build" PowerShell script that would build the application and create a Docker image on Windows Server 2016. To accomplish this task I wrote two functions, one to perform the msbuild and another to create the actual Docker image, as shown in **Figure 2**.

The next step in the script was to execute these two functions, passing in all required parameters:

Figure 3 **Output of the Build Script**

```
docker build -t myconsoleapplication .
Sending build context to Docker daemon  6.058MB
Step 1/3 : FROM microsoft/windowsservercore
 ---> 2cddde20d95d
Step 2/3 : ADD publish/ /
 ---> 452c4b42caa5
Removing intermediate container cafb387a3634
Step 3/3 : ENTRYPOINT myconsoleapplication.exe
 ---> Running in a128ff044ef3
 ---> 4c7dce888b36
Removing intermediate container a128ff044ef3
Successfully built 4c7dce888b36
Successfully tagged myconsoleapplication:latest
```



```
PS C:\Users\Administrator\src\myconsoleapplication\myconsoleapplication> docker images
REPOSITORY                     TAG                 IMAGE ID            CREATED             SIZE
myconsoleapplication           latest              4c7dce888b36        2 minutes ago       10.4GB
```

Figure 4 **Console Application As a Docker Image**

```
Invoke-MSBuild -MSBuildPath "MSBuild.exe" -MSBuildParameters
  ".\myconsoleapplication.csproj /p:OutputPath=.\publish
/p:Configuration=Release"
Invoke-Docker-Build -ImageName $ImageName -ImagePath "."
```

With the build script in hand, all that remained was to create my Docker file and my console application would be enabled for Windows Server Containers running on Windows Server 2016. Note that from a development standpoint, it can be helpful when testing the build process to use the Visual Studio command prompt, which will include MSBuild in your path. As part of the preliminary setup I installed a base Docker image named Windows Server Core that had all of the base features I needed to run my application. When creating the Docker file, I told Docker to use this image and publish my application with the name "myconsoleapplication.exe" as the entry point:

```
FROM microsoft/windowsservercore
ADD publish/ /
ENTRYPOINT myconsoleapplication.exe
```

The entry point will be the Main function in the console application.

> For my particular set of applications, I was reading in very large files and didn't want to copy the files to every container.

Final Build and Deployment to Windows Server 2016

Once I had a complete .NET console application that was enabled for Windows Server Containers, I was ready to deploy my application. An easy way I found to do this for testing was to simply copy the application folder to the VM. Once I copied the application to the server, I executed the PowerShell script to build the application. I navigated to the source directory and then ran the ./build command from PowerShell.

The output of the build script should look similar to the result shown in **Figure 3**.

To confirm that my Docker image was created successfully, I ran the docker images command again and I could see the new Docker Image, as shown in **Figure 4**.

Testing the Windows Server Container Console App

The very last step I took before getting into some very specific features was to test my application to make sure that it would, in fact, run within a Windows Server Container. To do so I ran the following command:

```
docker run --rm -it myconsoleapplication
  ".NET Framework App Running in Windows Container"
```

As expected, the application output the argument passed to it in the console window.

That took care of the basics for deploying, configuring and setting up a .NET app that can run in a Windows Server Container. At this point, you might be thinking of many existing applications you could potentially move to a Windows Server Container. However, there are still a few key features I found very helpful—such as file sharing and socket communication—that you might also find useful. In the next section I'll delve a little more into these features and how to leverage them in your own applications.

Docker Container: Enabling Advanced File Features

Like many console applications, yours may have a fair number of files that are being leveraged for different reasons; whether it's for logging or processing or something else, the use of files might be intensive. For my particular set of applications, I was reading in very large files and didn't want to copy the files to every container. I also wanted to do my best to optimize disk I/O, and using a shared folder on the network—a file server—introduced too much latency and impacted performance when trying to read in such large files. Furthermore, I didn't want to create multiple versions of my application with various configurations, ports and directories as this would be a maintenance nightmare. As a result, I started evaluating how I could share my files on my host system running the Docker service and then access those files from within my container. What I found is that this is extremely easy and it doesn't matter if you're using Windows Server Containers or running a Docker container in Linux. Docker has full support for this type of functionality. In fact, what was the most beneficial to my setup was that as long as I mounted a drive in the container to correlate with the internal directories, I didn't even need to modify my application. The only change I made was to have a parameter set my path when I ran the Docker container instead of reading it from a configuration file.

I was able to keep all of the file processing and paths intact because they were all relative paths and within a main directory. That meant I didn't have to change the core logic of my application, which in turn alleviated much of the risk I might have engendered by changing my .NET console application.

To test this functionality, I added a basic file IO process to my console application by inserting the following code:

```
using (StreamWriter sw = File.AppendText(@"C:\containertmp\testfile.txt"))
{
  sw.WriteLine(DateTime.Now.ToString() + " - " + args[0]);
}
```

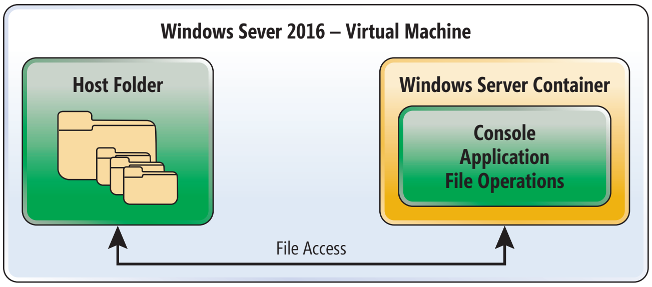
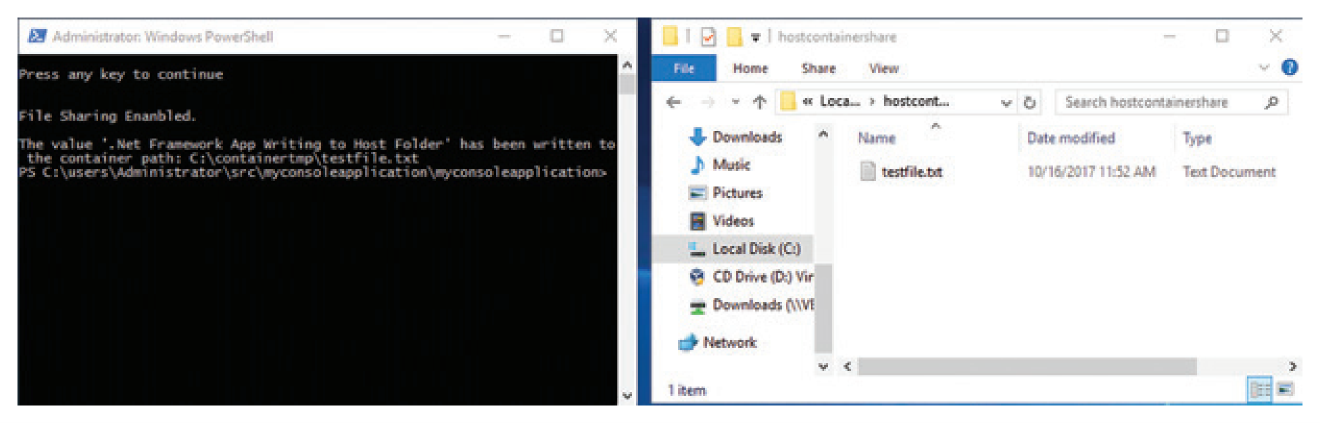

Windows Sever 2016 – Virtual Machine
Host Folder
Windows Server Container
Console Application File Operations
File Access

Figure 5 **Host and Windows Server Container File Operations Overview**



```
Press any key to continue


File Sharing Enanbled.

The value '.Net Framework App Writing to Host Folder' has been written to
 the container path: C:\containertmp\testfile.txt
PS C:\users\Administrator\src\myconsoleapplication\myconsoleapplication>
```

Figure 6 **Example of Write Access from the Docker Container to the Windows Server 2016 Host**

I then redeployed my solution to the Windows Server 2016 VM. This also required rebuilding the image by running the ./build PowerShell script.

The last step to enable this functionality was to create a directory on the host that I needed to expose to the Docker container. In my case, I just created a folder called hostcontainershare. The key to doing this was how I mounted this folder from the Windows Server Host System to the Docker container. Surprisingly, this is extremely easy to accomplish by passing in the following argument to the docker run command:

```
-v [source directory or path]:[container directory or path]
```

This argument is set up to accept a source and target. For example, I passed in first my local Windows Server Host directory and then how I wanted it mounted inside the container. Here's the entire docker run command:

```
docker run --rm -it -v c:\hostcontainershare:c:\containertmp myconsoleapplication
  ".NET Framework App Writing to Host Folder" 1
```

There are various ways to accomplish this functionality both in Windows Server Containers and Docker containers, but for my .NET console application, I found this method very simple and easy to implement. An illustration of how this is set up is shown in **Figure 5**.

The result of the docker run command was a file written to my host directory from within my Docker container, as shown in **Figure 6**.

Enabling this functionality provided significant advantages for my application because of what it's doing with very large files. I found I didn't need to duplicate my files across all of the containers. As long as I have optimal or solid state drives on my host, the file processing is much faster than using a shared folder, network drive or other non-local site. The benefits of using this technique are countless for traditional console applications.

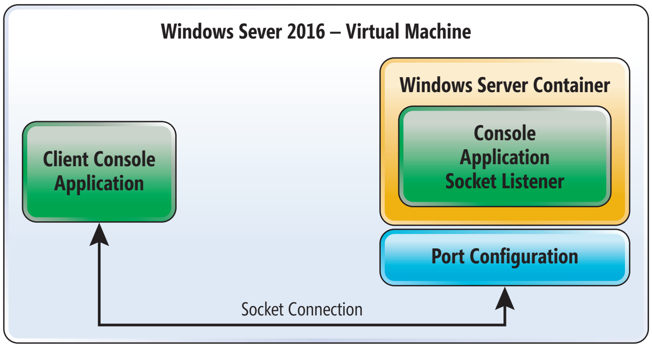

Windows Sever 2016 – Virtual Machine
Windows Server Container
Console Application Socket Listener
Client Console Application
Port Configuration
Socket Connection

Figure 7 **Client to Windows Server Container Host Socket Communication**

With successful file sharing, I had one last feature to conquer—socket connections, which I'll discuss in the next section.

> With successful file sharing, I had one last feature to conquer—socket connections.

Docker Container: Enabling Advanced Socket Features

One of the main features I needed to prove out was being able to communicate from a host socket connection to an internal container socket connection. Again, much of this functionality can be leveraged in both Windows Server Containers and Docker containers because the setup is controlled via command-line arguments that specify how the Docker container is running and what ports are being exposed.

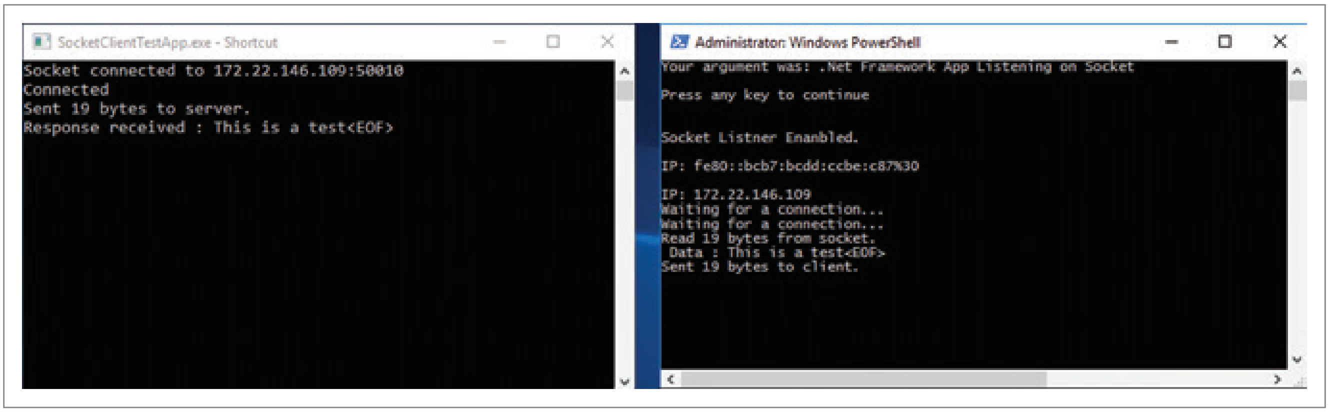
To support this functionality, I created client and server socket applications that would establish a connection from the client application running on Windows Server to a server-side application listener running as a Windows Server Container. I also added into my application the code necessary to listen on a specific socket and then respond in the console with the data and bytes received.

I leveraged Microsoft's socket examples from Asynchronous Client Socket Example at bit.ly/2gDKYz2 and Asynchronous Server Socket Example at bit.ly/2i8VUbK for the base code segments I integrated into my application.

I did make a few changes to the server-side code to assist with getting the IP address of the container so that when I was using the client socket application I'd be able to provide the assigned IP address. I was able to obtain the NAT details of the container by running the following command:

```
docker network inspect nat
```

I also ran various lookups to retrieve the IP address of the container, but to make it easy to debug and troubleshoot I added in a loop that retrieved all of the IP addresses and then wrote them out to the console window:


SocketClientTestApp.exe - Shortcut
Administrator: Windows PowerShell

```
Socket connected to 172.22.146.109:50010
Connected
Sent 19 bytes to server.
Response received : This is a test<EOF>
```

```
Your argument was: .Net Framework App Listening on Socket

Press any key to continue


Socket Listner Enanbled.

IP: fe80::bcb7:bcdd:ccbe:c87%30

IP: 172.22.146.109
Waiting for a connection...
Waiting for a connection...
Read 19 bytes from socket.
 Data : This is a test<EOF>
Sent 19 bytes to client.
```

Figure 8 **Client Socket Application Sending Data to Container Console Application**

```
foreach (var info in ipHostInfo.AddressList)
{
  Console.WriteLine("\nIP: " + info);
}
```

I also set the port to the specific port I was testing for my socket connection. I once again deployed my application to the Windows Server 2016 VM, as well as copied my client application to the server in order to test the connectivity. By default, no custom ports are exposed from the container and the container won't allow a TCP socket connection. In order to enable this functionality I needed to give Docker the appropriate run arguments, similar to what was needed to share a folder.

In my case, I wanted to connect to port 50020 from the host running my client application to the .NET console application running within my Windows Server Container. **Figure 7** illustrates how the application is set up.

Once everything was set up and configured, I needed to tell the Windows Server Container and Docker container that I want to expose certain ports from my container to the host machine. To enable this behavior I specified the following argument to the Docker run command:

```
-p [host port]:[container port]
```

You can expose multiple ports by repeating this argument for each one, for example -p 50020: 50020 –p 50019:50019, and so forth. By running my container and exposing the ports I was ready to test that I have a connection from the Windows Server Container console application to my client running on the Windows Server 2016 VM.

The complete command I used to run the Windows Server Container was:

```
docker run --rm -it -p 50010:50010 -v c:\hostcontainershare:c:\
containertmp myconsoleapplication
  ".NET Framework App Listening on Socket" 2
```

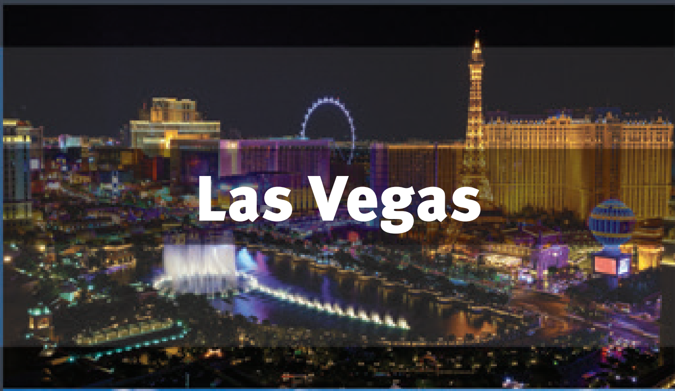
Once I launched the Windows Server Container running the console application, I was ready to start my client application. The container console application showed me the current IP address of the container and the fact that it was listening on the socket I specified. All I needed to do next was launch my client application and pass in the current IP address of the container I wanted the client application to connect to and my testing would be complete. As shown in **Figure 8**, the client application connected to the IP address of the container console application displayed on the screen and sent a small set of data over the socket. Success!

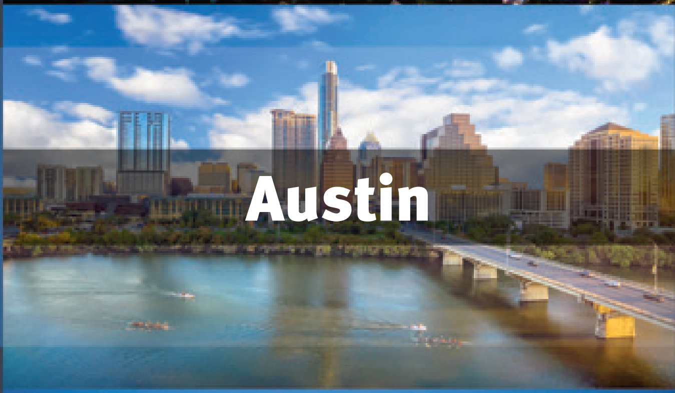
Wrapping Up

Given the nature of the applications I was running, I needed several specific features to be available with Docker. When I learned that Windows Server Containers would let me run my .NET console application, I was fairly optimistic that I'd be able to access a file from the host and enable socket communication from the host system to my Docker container. What I was most impressed with is the ability to share folders and files while also exposing sockets and ports specific to my applications or any other applications. With Windows Server 2016, the integration of Windows Server Containers is extremely smooth, with very little configuration or orchestration required to deploy Windows containers. For .NET app you're planning to migrate to Docker, I definitely recommend using Windows Server Containers and exposing features of Docker as needed to ensure you application will run as expected. As with most applications and sharing of resources, security must always be considered and reviewed. You still must use caution when sharing data or sockets from a host system to a container. When enabling such functionality, you need to be extremely careful not to introduce a vulnerability. In addition, sharing files and opening ports between a host system and a container must be handled with care to avoid security risks. I found that with my application, I was able to provide a high-level of scalability while also modernizing certain components of the application overall. The application can now be deployed into a more scalable setup using Docker Swarm or other scaling models that allow the application to run, limited only by cost or by the level of the hardware. As a bonus, this solution provided me with the much-needed time to evaluate if a redesign was needed or if this solution could be the permanent solution. With many of the features shown in this article, hopefully you can begin your own migration and design to modernize your .NET applications. ■

SEAN IANNUZZI *has been in the technology industry for more than 20 years and has played a pivotal role in bridging the gap between technology and business visions for a plethora of social networking, Big Data, database solutions, cloud computing, e-commerce and financial applications of today. Iannuzzi has experience with more than 50 unique technology platforms, has achieved more than a dozen technical awards/certifications and specializes in driving technology direction and solutions to help achieve business objectives.*

THANKS to the following Microsoft technical expert for reviewing this article: *Jesse Squire*


Las Vegas


Austin

Boston

Redmond

Chicago

San Diego

Orlando

The New Navigation View for UWP Apps

Jerry Nixon

The Windows XAML team released the NavigationView control with the Windows 10 Fall Creators Update. Prior to the control, developers tasked with implementing a hamburger menu were limited to the rudimentary features of the SplitView control. The resulting interfaces were inconsistent in both visual presentation and behavior. Even first-party Microsoft apps like Groove, Xbox, News and Mail struggled with visual alignment across the portfolio. Often an internal problem drives an external solution, as is the case here with NavigationView.

The control gives XAML developers a fresh and beautiful visual, consistently implemented across devices with comprehensive support for adaptive scaling, localization, accessibility, and signature Windows experiences like Microsoft's new Fluent design system. The control is beautiful, and end users are bound to lose endless hours of productivity just invoking the control's selection animation over and over. It's mesmerizing. **Figure 1** shows NavigationView basic styling.

The Basics

Adding the NavigationView to an app is simple. Generally, NavigationView is in a dedicated page, like ShellPage.xaml. Just a few lines of XAML give a menu, buttons and handlers to respond to typical user actions. See the NavigationViewItem in MenuItems in the following code:

```
<NavigationView SelectionChanged="SelectionChanged">
  <NavigationView.MenuItems>
    <NavigationViewItemHeader Content="Section A" />
    <NavigationViewItem Content="Item 01" />
    <NavigationViewItem Content="Item 02" />
  </NavigationView.MenuItems>
  <Frame x:Name="NavigationFrame" />
</NavigationView>
```

These are the primary navigation buttons. Other items supported are the NavigationViewItemHeader and NavigationViewItemSeparator, which, together, allow developers to compose beautiful and sophisticated menus. There are several things you should be thinking about as you work through the Navigation view.

Anatomy The parts of the NavigationView are as shown in **Figure 2**. Each area builds out a comprehensive UX. The Header, Pane Footer, Auto Suggest and Settings button are optional, depending on your app's design requirements.

Modes The control has three possible modes: Minimal, Compact and Extended, as depicted in **Figure 3**. Each is auto-selected based on a built-in and customizable view with thresholds. Modes allow the NavigationView to remain usable and practical as the size of the app or device changes.

Real-World Problems

The NavigationView is simple to understand, but not always easy to implement, especially within the context of sophisticated, real-world scenarios. Controls made accessible to every developer use case often require some clear-headed coding. Here's a rundown of issues and elements developers need to recognize. I'll be addressing each of these later in this article.

Data Binding Personally, I think it's ridiculous to data bind menu items to a top-level navigation control, but I realize not all developers agree. To that end, binding to the NavigationView from a codebehind is quite simple. Binding from a view-model requires breaking cardinal rules like referencing UI namespaces.

Navigating Developers might be surprised the NavigationView doesn't navigate. It's only a visual affordance for navigation. It doesn't have a Frame or understand what its menu items should do. The first thing developers will need to solve is simple navigation with some logic around reloading pages.

This article discusses:

- The basics of the NavigationView XAML control
- Using NavigationView to build an app with real-world requirements
- Control extensibility opportunities using standard techniques in XAML and C#

Technologies discussed:

Universal Windows Platform, XAML, Windows 10 Fall Creators Update, NavigationView control

Code download available at:

bit.ly/2AjgFVG

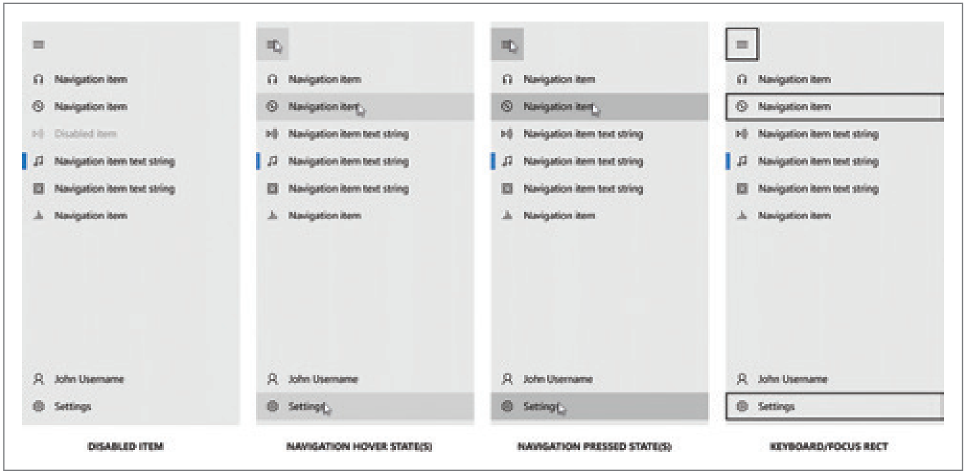


Figure 1 **NavigationView Basic Style**

Back Button Windows 10 provides a shell-drawn back button that's optional in some cases, but required in others, like tablet mode. The back button saves canvas real estate and establishes a unified point for back navigation. Attaching to the universal WinRT BackRequested event is straight forward, but synchronizing NavigationView's selection is another requirement.

Settings Button The NavigationView provides a default, localized Settings button at the bottom of the menu pane. It's brilliant. The button establishes a single, standard point of invocation for a common user action. It's the sort of thing designers and developers should learn from and adopt quickly for the sake of a visually aligned UX across the ecosystem.

Implementation of the Settings button is simple and clean, but it's another requirement of the NavigationView that's not delivered right out of the box. The problem lies in every XAML developer's desire to declare a control's behavior, rather than code it.

Header Items The NavigationView's MenuItem property accepts NavigationViewItemHeader objects used to visually bookend buttons; it's particularly useful to partition NavigationViewItems. But opening and closing the NavigationView's menu pane truncates the content of a header. Developers need to be able to control menu look and structure in both narrow and wide modes.

Real-World Solutions

XAML developers have several tools for solving problems. Inheriting from a control lets developers extend its behavior (bit.ly/2gQ4vN4), extension methods enhance the base implementation of even sealed controls (bit.ly/2ik1rfx) and attached properties can broaden the capabilities of a control (bit.ly/2giDGAn), even supporting declaration in XAML.

Data Binding Since 2006, when the XAML team invented it, Model-View-ViewModel (MVVM) has been the darling pattern of XAML developers, including Microsoft's own first-party apps. One principle of the design pattern is to prevent the reliance on and reference to UI namespaces in view-models. There are many reasons this is smart. As shown in the following code snippet, NavigationView supports the data binding of NavigationViewItems to the MenuItemsSource property, similar to ListView.ItemsSource, but it precludes UI namespaces. That's fine in codebehind, but a problem to solve for view-models:

```
public IEnumerable<object> MenuItems
{
  get
  {
    return new[]
    {
      new NavigationViewItem { Content = "Home" },
      new NavigationViewItem { Content = "Reports" },
      new NavigationViewItem { Content = "Calendar" },
    };
  }
}
```

To side-step referencing Windows.UI.Xaml.Controls in my view-model, I abstract the NavigationViewItem to a DTO. I repeat this process for each potential peer object. Every item's ordinal position is the responsibility of the view-model and should be maintained by the view logic. These abstractions are simple and easy for the view-model to provide, as shown in this code:

```
public class NavItemEx
{
  public string Icon { get; set; }
  public string Text { get; set; }
}

public class NavItemHeaderEx
{
  public string Text { get; set; }
}

public class NavItemSeparatorEx { }
```

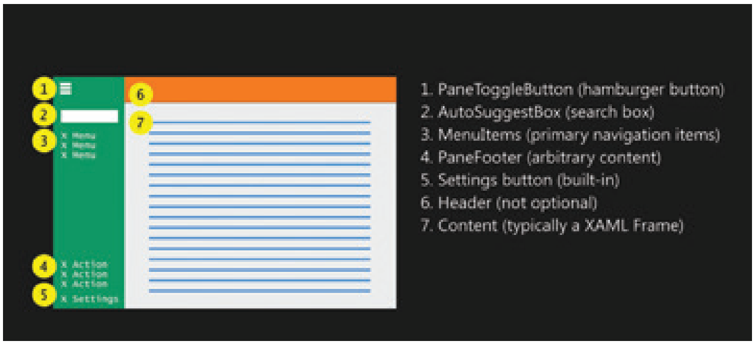

1
2
3
4
5
6
7
x Menu
x Menu
x Menu
x Action
x Action
x Action
x Settings
1. PaneToggleButton (hamburger button)
2. AutoSuggestBox (search box)
3. MenuItems (primary navigation items)
4. PaneFooter (arbitrary content)
5. Settings button (built-in)
6. Header (not optional)
7. Content (typically a XAML Frame)

Figure 2 **NavigationView Parts**

However, the NavigationView doesn't know my custom classes and they need to be converted to proper NavigationView controls for rendering. Binding to custom classes requires significant custom code in the NavigationView to coerce rendering, so we'll avoid this. Note: I am intentionally avoiding custom templates, so I don't mistakenly spoil accessibility or miss out on template improvements in subsequent platform releases. To make conversion easy, I introduce a value converter I can reference in my XAML binding. **Figure 4** shows the code responsible for taking my enumerable of custom classes and returning the objects that the NavigationView expects.

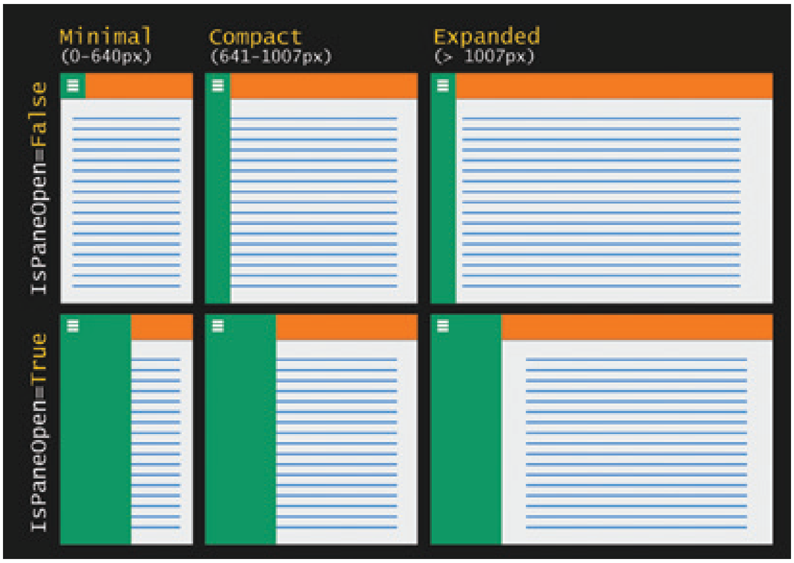


Figure 3 **NavigationView Modes**

After referencing this converter as an app-wide or page-level resource, the syntax is as simple as any other converter. I want to take a moment to reiterate how crazy I think it would be to data bind a top-level navigation, but this extensible solution works seamlessly, as shown here:

```
MenuItemsSource="{x:Bind ViewModel.Items, Converter={StaticResource NavConverter}}"
```

Navigating Navigation in the Universal Windows Platform (UWP) starts with the XAML Frame. But, the NavigationView doesn't have a Frame. In addition, there's no way to declare my intent with a menu button, which is to say, the page I want it to open. This is easily solved with the XAML attached properties shown here:

```
public partial class NavProperties : DependencyObject
{
  public static Type GetPageType(NavigationViewItem obj)
    => (Type)obj.GetValue(PageTypeProperty);
  public static void SetPageType(NavigationViewItem obj, Type value)
    => obj.SetValue(PageTypeProperty, value);
  public static readonly DependencyProperty PageTypeProperty =
    DependencyProperty.RegisterAttached("PageType", typeof(Type),
      typeof(NavProperties), new PropertyMetadata(null));
}
```

Once I have PageType on NavigationViewItem, I can declare the target page in XAML or bind it to my view-model. Note: I could add additional Parameter and TransitionInfo properties if my design required it; this sample focuses on a basic Navigation implementation. Then I let the extended NavigationView handle navigation, as shown in **Figure 5**.

Look at **Figure 5** and you'll notice four important enhancements. One, a XAML Frame is injected during control instantiation. Two, handlers have been added for Frame.Navigated, ItemInvoked and BackRequested. Three, SelectedItem has been overridden to add BackStack and BackButton logic. And four, a new SettingsPageType property has been added to the class.

Back Button The new, explicit frame isn't just a convenience, it gives me the source for Navigation events. This is important. When the NavigationView invokes navigation, I update the visibility of the shell-drawn back button. Should the user navigate another way, however, I can't know to update the back button without some sort of event. The Frame.Navigated event is an excellent, global choice.

Find An unexpected behavior of the NavigationView's ItemInvoked event is that the InvokedItem property passed in the custom event arguments is the string content of the NavigationViewItem and not an object reference to the item itself. As a result, the Find methods in this customized control locate the correct NavigationViewItem based on content passed in ItemInvoked or PageType passed in the Frame.Navigated event.

It's worth noticing the content of NavigationViewItem can change dynamically with localization settings on the device. Handling ItemInvoked with a hardcoded switch statement, as demonstrated in the online documentation (bit.ly/2xQodCM) would work for English-speakers only or require the switch to exponentially expand as languages are added to support a UWP app. Try to avoid magic numbers and magic strings anywhere in your code. They're not compatible with significant code bases.

Settings The settings button is the only button in the lower menu pane that participates in the selection logic of the NavigationView. Invoking it, users navigate to the settings page. To simplify that implementation, notice the custom SettingsPageType property, which holds the desired target page type for settings.

Figure 4 **Converter for NavItems**

```
public class INavConverter : IvalueConverter
{
  public object Convert(object v, Type t, object p, string l)
  {
    var list = new List<object>();
    foreach (var item in (v as Ienumerable<object>))
    {
      switch (item)
      {
        case NavItemEx dto:
          list.Add(ToItem(dto));
          break;
        case NavItemHeaderEx dto:
          list.Add(ToItem(dto));
          break;
        case NavItemSeparatorEx dto:
          list.Add(ToItem(dto));
          break;
      }
    }
    return list;
  }

  object IvalueConverter.ConvertBack(object v, Type t, object p, string l)
    throw new NotImplementedException();

  NavigationViewItem ToItem(NavItemEx item)
    new NavigationViewItem
    {
      Content = item.Text,
      Icon = ToFontIcon(item.Icon),
    };

  FontIcon ToFontIcon(string glyph)
    new FontIcon { Glyph = glyph, };

  NavigationViewItemHeader ToItem(NavItemHeaderEx item)
    new NavigationViewItemHeader { Content = item.Text, };

  NavigationViewItemSeparator ToItem(NavItemSeparatorEx item)
    new NavigationViewItemSeparator { };
}
```

The overridden SelectedItem setter tests for the settings button and consequently navigates as declared.

What isn't handled in either the NavigationViewItem's PageType property or the SettingsPageType property is a way to indicate custom TransitionInfo to the Frame's Navigate method to coerce the transition information during navigation. This can be an important customization to any app, and additional custom or attached properties could be added to allow for this additional instruction. The code to accomplish this looks like this:

```
<local:NavViewEx SettingsPageType="views:SettingsPage">
  <NavigationView.MenuItems>
    <NavigationViewItem Content="Item 01"
      local:NavProperties.PageType="views:Page01" />
    <NavigationViewItem Content="Item 02"
      local:NavProperties.PageType="views:Page02" />
    <NavigationViewItem Content="Item 03"
      local:NavProperties.PageType="views:Page03" />
  </NavigationView.MenuItems>
</local:NavViewEx>
```

Figure 5 **NavViewEx, an Extended NavigationView**

```
public class NavViewEx : NavigationView
{
  Frame _frame;

  public Type SettingsPageType { get; set; }

  public NavViewEx()
  {
    Content = _frame = new Frame();
    _frame.Navigated += Frame_Navigated;
    ItemInvoked += NavViewEx_ItemInvoked;
    SystemNavigationManager.GetForCurrentView()
      .BackRequested += ShellPage_BackRequested;
  }

  private void NavViewEx_ItemInvoked(NavigationView sender,
    NavigationViewItemInvokedEventArgs args)
  {
    if (args.IsSettingsInvoked)
      SelectedItem = SettingsItem;
    else
      SelectedItem = Find(args.InvokedItem.ToString());
  }

  private void Frame_Navigated(object sender, NavigationEventArgs e)
    => SelectedItem = (e.SourcePageType == SettingsPageType)
      ? SettingsItem : Find(e.SourcePageType) ?? base.SelectedItem;

  private void ShellPage_BackRequested(object sender, BackRequestedEventArgs e)
    => _frame.GoBack();

  NavigationViewItem Find(string content)
    => MenuItems.OfType<NavigationViewItem>()
      .SingleOrDefault(x => x.Content.Equals(content));

  NavigationViewItem Find(Type type)
    => MenuItems.OfType<NavigationViewItem>()
      .SingleOrDefault(x => type.Equals(x.GetValue(NavProperties.
PageTypeProperty)));

  public virtual void Navigate(Frame frame, Type type)
    => frame.Navigate(type);

  public new object SelectedItem
  {
    set
    {
      if (value == SettingsItem)
      {
        Navigate(_frame, SettingsPageType);
        base.SelectedItem = value;
        _frame.BackStack.Clear();
      }
      else if (value is NavigationViewItem i && i != null)
      {
        Navigate(_frame, i.GetValue(NavProperties.PageTypeProperty) as Type);
        base.SelectedItem = value;
        _frame.BackStack.Clear();
      }
      UpdateBackButton();
      }
  }

  private void UpdateBackButton()
  {
    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
      (_frame.CanGoBack) ? AppViewBackButtonVisibility.Visible
        : AppViewBackButtonVisibility.Collapsed;
  }
}
```

This kind of extensibility allows developers to aggressively extend the behavior of controls and classes without altering their fundamental, underlying implementations. It's a capability of C# and XAML that has been there for years and makes the coding syntax terse and the XAML declaration plain. It's an intuitive approach that translates to other developers clearly with little instruction.

Start Page When an app loads, no menu item is initially invoked. Adding another attached property, as shown below, lets me declare my intent in XAML so the extended NavigationView can initialize the first page in its Frame. Here's the property:

```
public partial class NavProperties : DependencyObject
{
  public static bool GetIsStartPage(NavigationViewItem obj)
    => (bool)obj.GetValue(IsStartPageProperty);
  public static void SetIsStartPage(NavigationViewItem obj, bool value)
    => obj.SetValue(IsStartPageProperty, value);
  public static readonly DependencyProperty IsStartPageProperty =
    DependencyProperty.RegisterAttached("IsStartPage", typeof(bool),
      typeof(NavProperties), new PropertyMetadata(false));
}
```

Using this new property in the NavigationView is a matter of locating the NavigationViewItem within MenuItems with the Start property set, then navigating to it when the control has successfully loaded. This logic is optional, supporting the setting but not requiring it, as shown here:

```
Loaded += (s, e) =>
{
  if (FindStart() is NavigationViewItem i && i != null)
    Navigate(_frame, i.GetValue(NavProperties.PageTypeProperty) as Type);
};

NavigationViewItem FindStart()
  => MenuItems.OfType<NavigationViewItem>()
    .SingleOrDefault(x => (bool)x.GetValue(NavProperties.IsStartPageProperty));
```

Notice the use of the LINQ SingleOrDefault selector in my FindStart method, as opposed to its selector sibling, First. Where FirstOrDefault returns the first found, SingleOrDefault throws an exception should more than one be discovered by its predicate. This helps guide and even enforce developer usage of the property, because only one initial page should ever be declared.

Page Header As shown in **Figure 2**, the NavigationView Header isn't optional. This area above the Page, with a fixed height of 48px, is intended for global content. Implementing a simple title is as easy as the snippet here, which attached a Header property to the Page object:

```
public partial class NavProperties : DependencyObject
{
  public static string GetHeader(Page obj)
    => (string)obj.GetValue(HeaderProperty);
  public static void SetHeader(Page obj, string value)
    => obj.SetValue(HeaderProperty, value);
  public static readonly DependencyProperty HeaderProperty =
    DependencyProperty.RegisterAttached("Header", typeof(string),
      typeof(NavProperties), new PropertyMetadata(null));
}
```

Figure 6 **Updating the Header**

```
private void Frame_Navigated(object sender,
  Windows.UI.Xaml.Navigation.NavigationEventArgs e)
{
  SelectedItem = Find(e.SourcePageType);
  UpdateHeader();
}

private void UpdateHeader()
{
  if (_frame.Content is Page p
    && p.GetValue(NavProperties.HeaderProperty) is string s
    && !string.IsNullOrEmpty(s))
  {
    Header = s;
  }
}
```

With the Frame's Navigated event, NavViewEx looks for the property in the resulting page, injecting the optional value into the NavigationView's Header. The new attached Page property can be scoped to individual pages and localized through the UWP x:Uid localization subsystem. The code in **Figure 6** shows how updating the header effectively introduces only two new lines of code to the extended control.

In this simple example the default TextBlock in the Header is accepted. In my experience, and corroborated by Microsoft's first-party, in-box apps, a CommandBar control typically takes up this valuable screen real estate. If I wanted the same in my app, I could update the HeaderTemplate property with this simple markup:

```
<NavigationView.HeaderTemplate>
  <DataTemplate>
    <CommandBar>
      <CommandBar.Content>
        <Grid Margin="12,5,0,11" VerticalAlignment="Stretch">
          <TextBlock Text="{Binding}"
            Style="{StaticResource TitleTextBlockStyle}"
            TextWrapping="NoWrap" VerticalAlignment="Bottom"/>
        </Grid>
      </CommandBar.Content>
    </CommandBar>
  </DataTemplate>
</NavigationView.HeaderTemplate>
```

That TextBlock styling mimics the control's default Header, placing it inside a globally available CommandBar, which can programmatically be implemented by an app on a page-by-page or global context. As a result, the basic design is visually the same, but its functional potential is significantly expanded.

The Narrow Item Header Issue

One problem remains. As described early in this article, the NavigationView has different display modes that vary based on view width. It also can explicitly open and close the menu pane. When the menu pane is open, its width is determined by the value of the OpenPaneLength property. Don't get me started on that property name using Length instead of Width. Anyway, here's the important part: That property value doesn't impact the width of the menu pane when it's closed; in the closed state, the pane width is hardcoded to 48px wide.

Here, NavigationViewItems look great with their icons set to 48px wide, but NavigationViewItemHeaders have only one Content property, and it's the same if the pane is open or closed. Attractive when open, the text is truncated when closed, as shown in **Figure 7**.

What to do? I first thought of adding an Icon to headers, but when the pane is closed it would look like a NavigationViewItem, but with the bizarre and possibly frustrating behavior of not responding to taps. I thought about alternate text, but inside 48px there's barely room for three characters. I finally landed on hiding headers when the pane is closed, as shown in the following code snippet:

```
RegisterPropertyChangedCallback(IsPaneOpenProperty, IsPaneOpenChanged);
private void IsPaneOpenChanged(DependencyObject sender,
  DependencyProperty dp)
{
  foreach (var item in MenuItems.OfType<NavigationViewItemHeader>())
  {
    item.Opacity = IsPaneOpen ? 1: 0;
  }
}
```

In this case, changing its visibility prevented any sudden movement of the items in the list. This is not only the easiest to implement, it's also visually pleasant, and somewhat intuitive as to why it's occurring. Because NavigationView doesn't expose an Opened or Closed event, you register for a dependency property change on the IsPaneOpenProperty using RegisterPropertyChangedCallback, a handy utility introduced with Windows 8. I'll identify the callback and toggle every header. If I wanted to, I could treat different headers in different ways; this example handles all headers the same.

Wrapping Up

What's beautiful about the Universal Windows Platform and XAML is the abundance of solutions to problems. No control meets the needs of every developer. No API meets the needs of every design. Building on a rich platform with extensive love for its developers turns snags into solutions with just a drop of code and a little effort. It lets you create your own signature experiences with unique value propositions that set your app apart in the ecosystem. Now, even the hamburger menu is a simple addition to your look-and-feel with opportunities for extensions around every corner. ■

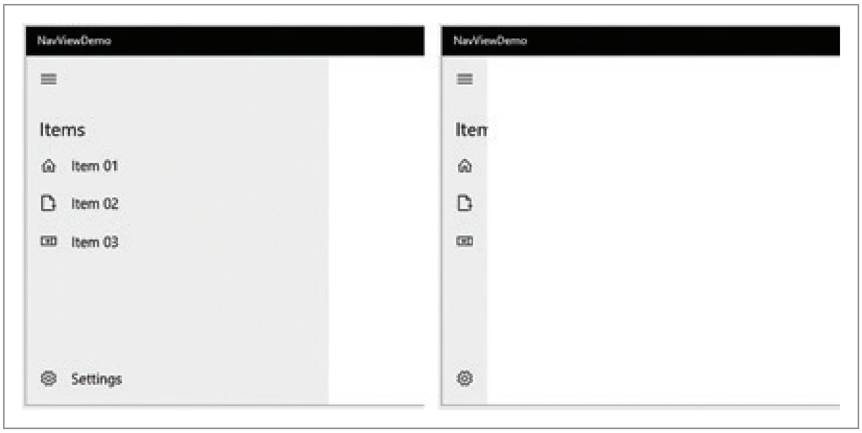


Figure 7 **NavigationViewHeader in Open and (Narrow) Closed States**

JERRY NIXON *is an author, speaker, developer and evangelist in Colorado. He trains and inspires developers around the world to build better apps with crafted code. Nixon spends the bulk of his free time teaching his three daughters Star Trek character backstories and episode plots.*

THANKS to the following technical expert for reviewing this article: *Daren May*

VBITS'99
VSLive! 1999

eturn this.http.get<IMo
VSLive! 2017

Customizing Visual Studio for Mac

Alessandro Del Sole

Visual Studio for Mac is a fully featured native development environment designed for building cross-platform applications with Xamarin and .NET Core on macOS. The IDE enables productivity through a rich set of features and tools, combined with powerful customizations that allow developers to implement their preferences. In my previous article (`msdn.com/magazine/mt845621`), I discussed Visual Studio for Mac productivity from the point of view of the code editor and debugging tools. In this article, I'll focus on the customization points that Visual Studio for Mac offers, and the impact those points have on productivity. Most of the customizations described in this article can be done in the Preferences dialog, which you open by selecting the Visual Studio Preferences menu.

Applying Languages and Themes

Visual Studio for Mac quickly allows you to change the display language for the UI. You can do this in the Visual Style tab of the Preferences dialog, selecting one of the available languages from the User Interface Language combo box. Currently, Visual Studio for Mac supports the following languages: Chinese (China and Taiwan), Czech, French, German, Italian, Japanese, Korean, Polish, Portuguese (Brazil), Russian, Spanish, Turkish and, of course, English.

In the same tab, you can change the appearance of the Visual Studio for Mac UI with different themes. At the time of this writing, Visual Studio for Mac offers two themes: Light and Dark. You can apply a theme by selecting the Visual Style tab in the Preferences dialog, and then select a theme from the User Interface Theme dropdown.

When you select a different theme, you'll be invited to restart Visual Studio. At restart, you'll see how the theme affects not only the code editor, but the entire workspace, including the pads and dialogs. **Figure 1** shows an example based on the Dark theme.

You can actually change the theme for the code editor only, rather than for the complete workspace. This can be useful to keep the editor window highlighted. The Color Theme tab in the Preferences dialog allows you to select from a long list of built-in themes, as well as any themes you create and import on your own. Visual Studio for Mac supports the Visual Studio (.vssettings), Xamarin Studio (.json) and TextMate (.tmTheme) formats, so you can quickly import new themes by simply pressing the Add button and specifying one or more supported themes.

Customizing Keyboard Shortcuts

Visual Studio for Mac offers a huge number of predefined keyboard shortcuts, referred to as key bindings, that make quick work of invoking common commands by reducing time spent mousing around the interface. Key bindings are completely customizable,

This article discusses:

- Customizing keyboard shortcuts
- Customizing Fonts
- Applying themes and custom layouts
- Extensions and custom commands

Technologies discussed:

Visual Studio for Mac, Visual Studio Extensions and you can change them in the Key Bindings tab of the preferences dialog. As you can see in **Figure 2**, key bindings are grouped by menu. You can change a key binding by clicking a command and then entering the new binding in the Edit Binding textbox.

Visual Studio for Mac provides a number of built-in schemes with key bindings that recall the same shortcuts used in other popular development tools, such as Visual Studio Code and Xcode. For example, if you've been building apps with Xcode for a while, you can select the Xcode scheme and use the same keyboard shortcuts in Xcode for the code editor and debugging in Visual Studio for Mac. As another example, if you've been working with Visual Studio on Windows and now you need to work with Visual Studio for Mac, you can select the Visual Studio (Windows) scheme to make use of its familiar keyboard shortcuts.

Customizing Fonts

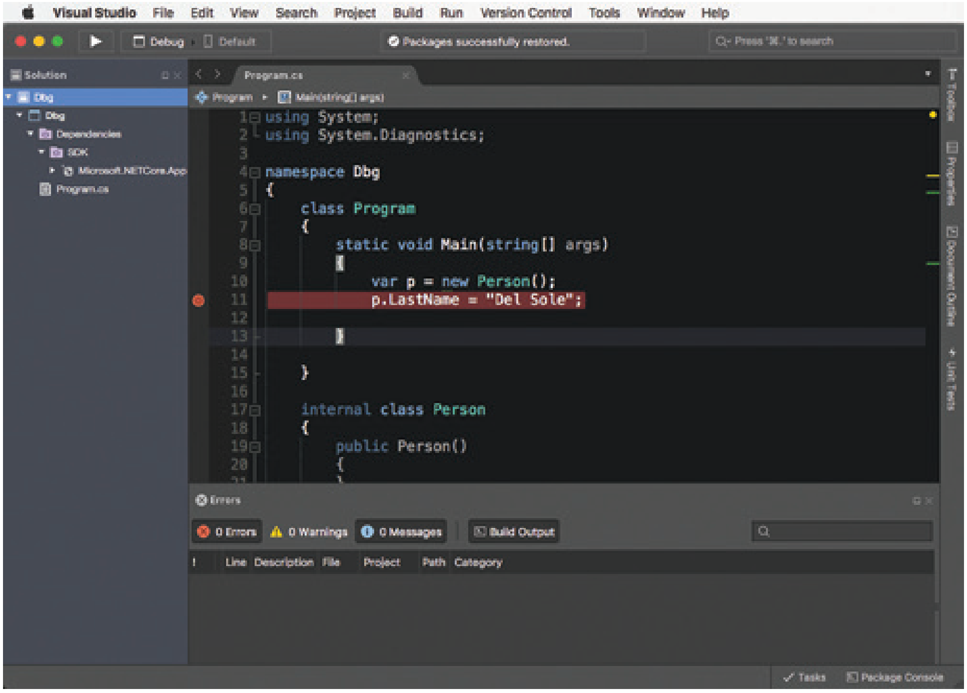
Visual Studio for Mac gives you control over the fonts used in the code editor, the Output pad, and all other pads (pads are tool windows that can be rearranged and docked in the workspace). Just click the Fonts tab in the Preferences dialog box to access this functionality.

To make a font change, select the font you want to replace, then click the new font (with style and size) in the Select Font dialog. A preview window lets you see what the new font selection will look like. Note that you can replace a default font with the newly selected font via the Set To Default button.

Adding Custom Commands

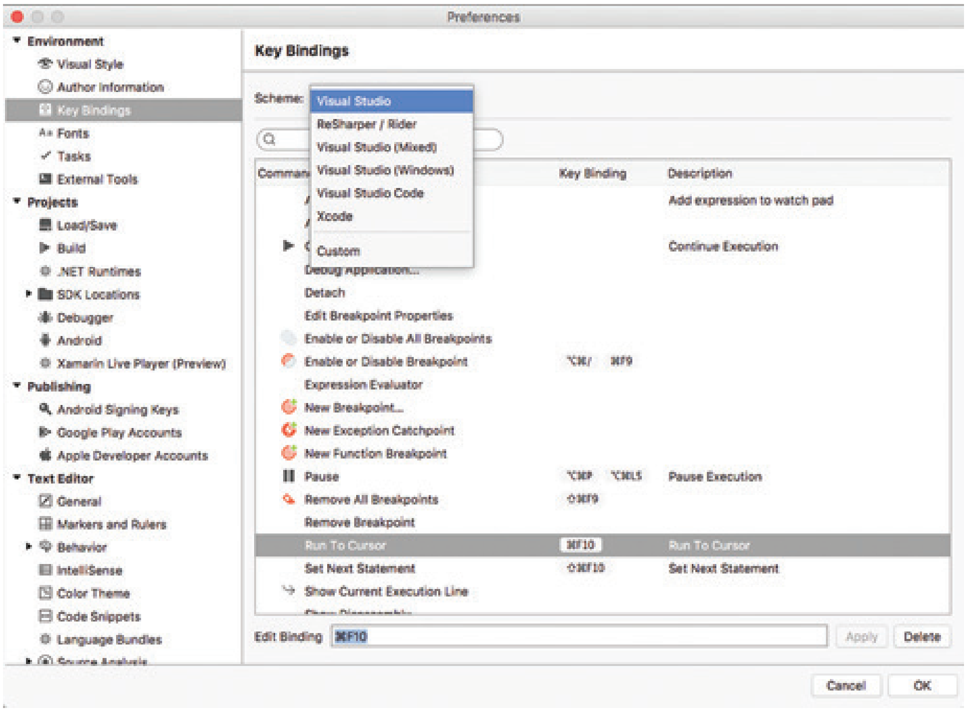
A common need for developers is to be able to launch external tools from within the development environment against one or more files in the solution. For example, you might want to launch a professional image-editing tool on a bitmap you have in a project, or you might want to launch a particular code editor or command line against a file in a project. Visual Studio for Mac allows you to launch external tools by adding new custom commands to the Tools menu.

To accomplish this, open the External Tools tab in the Preferences dialog and then click Add. In the screen that appears, provide the details for the external tool, such as the text you want to be displayed in the Tools menu (Title field), the tool to be launched (Command field), command arguments such as file or folder names (Arguments field), the directory where the command must be executed (Working directory field) and a keyboard shortcut (Key Binding field). **Figure 3** shows an example that launches Visual Studio Code.



```
using System;
using System.Diagnostics;

namespace Dbg
{
    class Program
    {
        static void Main(string[] args)
        {
            var p = new Person();
            p.LastName = "Del Sole";

        }

    }

    internal class Person
    {
        public Person()
        {
```

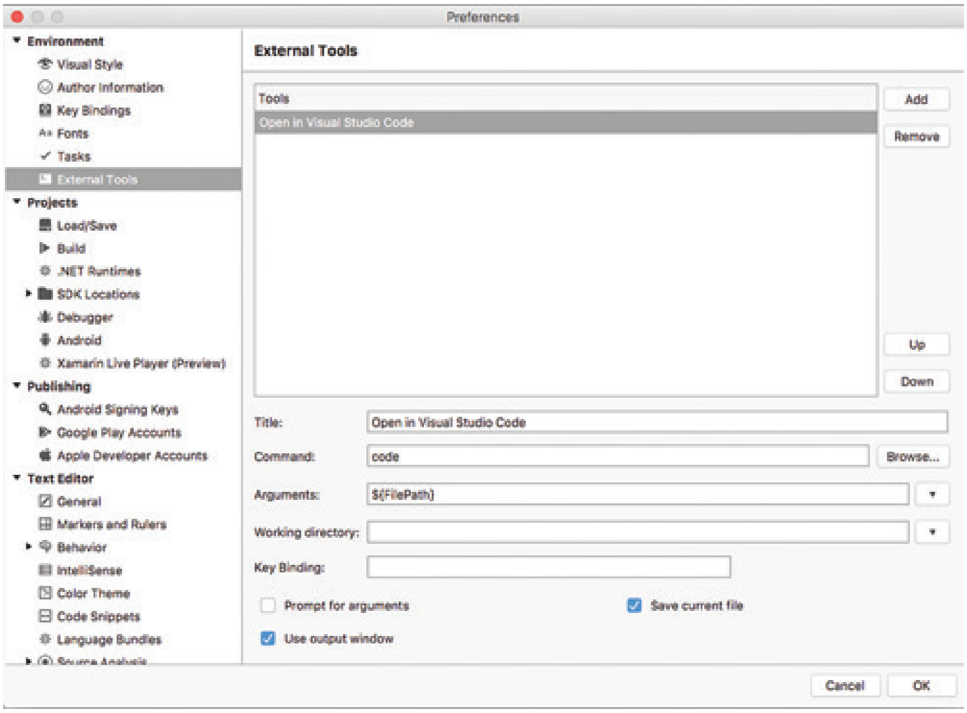
Figure 1 **Visual Studio for Mac with Dark Theme**



Command	Key Binding	Description
		Add expression to watch pad
		Continue Execution
Debug Application...		
Detach		
Edit Breakpoint Properties		
Enable or Disable All Breakpoints		
Enable or Disable Breakpoint	⌥⌘/ ⌘F9	
Expression Evaluator		
New Breakpoint...		
New Exception Catchpoint		
New Function Breakpoint		
Pause	⌥⌘P ⌥⌘L5	Pause Execution
Remove All Breakpoints	⇧⌘F9	
Remove Breakpoint		
Run To Cursor	⌘F10	Run To Cursor
Set Next Statement	⇧⌘F10	Set Next Statement
Show Current Execution Line		

Edit Binding ⌘F10

Figure 2 **Customizing Key Bindings**



Open in Visual Studio Code

Title: Open in Visual Studio Code

Command: code

Arguments: ${FilePath}

Working directory:

Key Binding:

☐ Prompt for arguments ☑ Save current file

☑ Use output window

Figure 3 **Adding a Command to Invoke an External Tool**

Notice that, for the Arguments and Working directory fields, the target pathname must be supplied via one of the supported constants, such as ${FilePath} that represents the pathname of the current file. However, it isn't necessary to remember all the possible constants and their meaning. In fact, you can click the arrow-down button at the right of both fields to select a target, and Visual Studio for Mac will show a human-readable description for each target, then will place the corresponding constant into the field when you select the desired target.

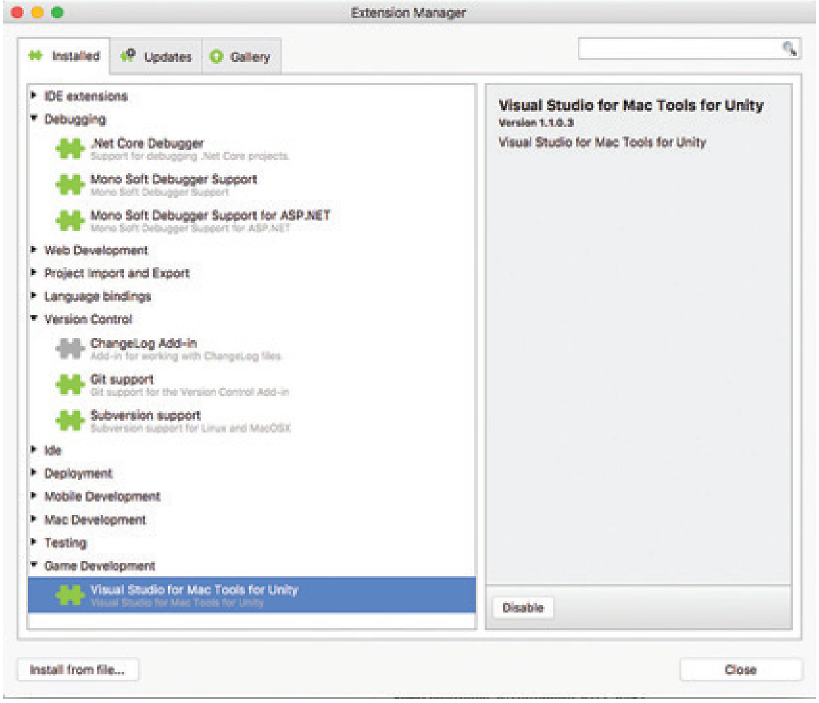
With regard to **Figure 3**, you'll see File Path in the list of possible targets (which represents the current file). When you select this option, Visual Studio for Mac will add the ${FilePath} constant into the Arguments field. You can also select multiple targets in one field.

Finally, it's worth mentioning the checkboxes at the bottom of the dialog. If selected, Prompt for arguments will cause Visual Studio to ask you to enter additional arguments that will be passed to the external tool. When the Save current file checkbox is selected, the target file will be saved before the external tool is launched. By checking the Use output window checkbox, the output of the external tool will be redirected to the Output pad in Visual Studio, which is extremely convenient so that you won't need to shift your focus outside of the IDE. After you click OK, you'll see a new command in the Tools menu, the text of which exactly matches the string you entered in the Title field.

Working with Custom Layouts

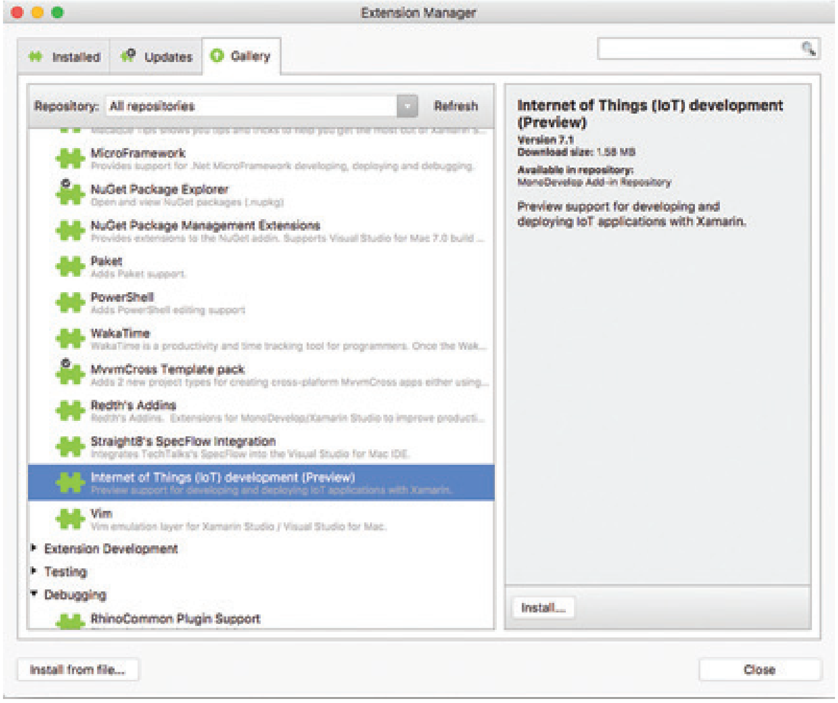
You can arrange the layout of Visual Studio for Mac by displaying or hiding some pads, and by moving and docking pads to a different position in the workspace. The IDE ships with four built-in, general-purpose layouts called Code, Design, Debug and Test, all available in the View menu, and each quickly allowing you to switch to a different pad layout depending on the context. For example, when you start debugging an application, Visual Studio for Mac automatically switches to the Debug layout and then goes back to the previous layout, usually Code or Design, once you're finished. However, it is common to rearrange the IDE layout based on the developer's preferences or on the type of the solution with which the developer is working. For example, when working with Xamarin solutions, you might need specific pads that you don't use with ASP.NET Core solutions and vice versa, or you might want to organize pads in a way that's similar to Visual Studio on Windows.

Instead of manually rearranging pads every time, Visual Studio for Mac lets you save your own layouts with the Save Current Layout command in the View menu. This command asks you to enter the name of the new layout, then stores your current layout and adds its name in the View menu, below the names of the built-in layouts. This way you can quickly


Extension Manager
Installed
Updates
Gallery
IDE extensions
Debugging
.Net Core Debugger
Support for debugging .Net Core projects.
Mono Soft Debugger Support
Mono Soft Debugger Support
Mono Soft Debugger Support for ASP.NET
Mono Soft Debugger Support for ASP.NET
Web Development
Project Import and Export
Language bindings
Version Control
ChangeLog Add-in
Add-in for working with ChangeLog files
Git support
Git support for the Version Control Add-in
Subversion support
Subversion support for Linux and MacOSX
Ide
Deployment
Mobile Development
Mac Development
Testing
Game Development
Visual Studio for Mac Tools for Unity
Visual Studio for Mac Tools for Unity
Disable
Install from file...
Close

Visual Studio for Mac Tools for Unity
Version 1.1.0.3
Visual Studio for Mac Tools for Unity

Figure 4 **Displaying the List of Installed Extensions**



MicroFramework
Provides support for .Net MicroFramework developing, deploying and debugging.
NuGet Package Explorer
Open and view NuGet packages (.nupkg)
NuGet Package Management Extensions
Provides extensions to the NuGet addin. Supports Visual Studio for Mac 7.0 build ...
Paket
Adds Paket support.
PowerShell
Adds PowerShell editing support
WakaTime
WakaTime is a productivity and time tracking tool for programmers. Once the Wak...
MvvmCross Template pack
Adds 2 new project types for creating cross-platform MvvmCross apps either using...
Redth's Addins
Redth's Addins. Extensions for MonoDevelop/Xamarin Studio to improve producti...
Straight8's SpecFlow Integration
Integrates TechTalk's SpecFlow into the Visual Studio for Mac IDE.
Internet of Things (IoT) development (Preview)
Preview support for developing and deploying IoT applications with Xamarin.
Vim
Vim emulation layer for Xamarin Studio / Visual Studio for Mac.
RhinoCommon Plugin Support

Internet of Things (IoT) development (Preview)
Version 7.1
Download size: 1.58 MB
Available in repository:
MonoDevelop Add-in Repository

Preview support for developing and deploying IoT applications with Xamarin.

Figure 5 **Discovering and Installing Extensions from the Online Gallery**

switch to your favorite layout with a single click. Once you select a custom layout, the View menu also enables the Delete Current Layout command for removing custom layout from the list. As you might expect, this command is disabled for built-in layouts.

Extending Visual Studio

Visual Studio for Mac is an environment built upon modules that expose a number of extensibility points. This means that other modules can be installed and that the IDE can be extended with third-party packages, referred to as extensions. This allows you to add new productivity features and tools to Visual Studio. You install, update and manage extensions in the Extension Manager dialog, which you enable with the Extensions command in the Visual Studio menu. In the Installed tab (see **Figure 4**), you can see the list of installed extensions grouped by category.

You can select an extension and see detailed information in the window to the right. You also have the option to disable or uninstall an extension.

Some built-in, integrated tools in Visual Studio for Mac are extensions themselves. For these extensions, the Uninstall button is generally unavailable and the Disable button is only available if disabling the extension doesn't affect the core features of the IDE. Extensions that are part of the core of Visual Studio for Mac, such as the ChangeLog Add-in in **Figure 4**, are presented in gray in the Extension Manager.

If you switch to the Gallery tab, you'll see a list of available extensions from an online gallery, as shown in **Figure 5**. In the Repository combo box you can choose to see only stable extensions, only beta release extensions or all extensions.

When you've found an extension of interest, click its name on the left, review the details on the right and then click Install. Visual Studio will ask for confirmation and then install the selected extension. Depending on the extension, you'll find the IDE updated with new project templates, new menu commands, new pads or new context commands. The description you get in the Extension Manager tool should clarify how you access the new tools.

It's worth mentioning that you can also develop your own extensions for Visual Studio for Mac using an extension called Add-in Maker, which you can find under the Extension Development group of the Gallery tab in the Extension Manager. This package will install all the tools you need to build extensions, including specific project templates that appear in the Miscellaneous node of the New Project dialog.

Add-in Maker is an open source project (bit.ly/2zKxWIa) that's already emerged as the tool of choice for building extensions for IDEs such as Xamarin Studio and MonoDevelop. The official Microsoft documentation provides an interesting page that explains the extensibility points in Visual Studio for Mac (bit.ly/2yIpvNn) and a walk-through that provides an example based on a simplified extension that makes it possible to insert the current date in the active editor through a command added to the Edit menu (bit.ly/2yJ1V4E). If you plan to build extensions for Visual Studio for Mac with Add-in Maker, I strongly recommend you read these documents before you get started.

Wrapping Up

Productivity often depends on how comfortable the developer feels with an IDE. Visual Studio for Mac puts productivity at its core, enabling developers to customize the most important areas of the workspace. As a developer, you can localize the UI by selecting from a list of available cultures. You can change the visual theme to get the colors you like most, and you can customize fonts according to your preferences. You can also rearrange the pads layout and save each layout for later use, so that you won't need to manually rearrange pads every time.

Of course, Visual Studio for Mac can be enhanced with custom commands to invoke external tools, which is a very common need. And it can be extended with third-party packages that make it easier to add new features and tools. Combined, these capabilities enable developers to feel more comfortable with the IDE and to customize it in a way to maximize productivity. ■

ALESSANDRO DEL SOLE *has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos and articles about .NET development with Visual Studio. Del Sole works as a senior .NET Developer, focusing on .NET and mobile app development, training and consulting. He has recently authored an upcoming book called "Beginning Visual Studio for Mac" (bit.ly/2hsRxYx). You can follow him on Twitter: @progalex.*

THANKS to the following Microsoft technical expert for reviewing this article: *Jordan Matthiesen*

Using Cognitive Services in Mixed Reality

Tim Kulp

Mixed reality (MR) has a lot of potential for enhancing the way users interact with their environment. You can build immersive worlds for them to explore, or you can extend the real world with digital objects. Whether creating next-generation games, education tools or business apps, MR lets you build new ways of interacting with the digital world. In this article I'm going to show you how to take your MR app further into the real world—away from onscreen buttons and to a conversation-based interface—by using Microsoft Cognitive Services.

Cognitive Services is a powerful complement to MR. One of the limits of MR is its comprehension of the world. Tools like Spatial Mapping provide the MR system with an understanding of surfaces and collision points (where your digital content can rest but not pass through). Cognitive Services allows you to turn surfaces into tables, chairs, flooring and more. In other words, Cognitive Services lets MR understand that a particular surface is not just a surface, but a wooden table. Beyond understanding the environment, Cognitive Services enables MR to further reflect the physical world by using speech for interactions with digital objects, and custom gestures to tailor how people think about interacting with objects.

This article discusses:

- Building a mixed reality (MR) question-and-answer app for field technicians
- Creating a target-based MR scene
- Adding conversational interaction to the app

Technologies discussed:

Unity, Vufonia, Microsoft Cognitive Services, HoloLens, C#

Code download available at:

bit.ly/2gF8LhP

> Cognitive Services is a powerful complement to MR.

For this article, I'm going to build an MR app using Unity, C# and Vuforia. You'll want to start with Unity Beta 2017.3.0b3, which has Vuforia as an installation option. Make sure to install the Vuforia components for this version of Unity. To use Vuforia, you'll need to sign up for the Developer Portal and set up your targets there. Details about how to set up your Vuforia account can be found at bit.ly/2yrE6yH. Keep in mind that because you'll be using Unity, you'll be working with the Microsoft .NET Framework 3.5.

What Am I Building?

I'll be building an app for Contoso Power, an energy utility that is using MR for field services. The application will provide answers to technicians in the field based on the equipment the technician scans for the MR experience. Imagine that each piece of equipment has unique repair instructions. When the technician scans the MR marker on the equipment, the system identifies the question/answer set to use. The technician can then verbally activate the app's "listening mode," ask the question (again verbally) and get an audio response from the app. When the technician is finished, they can end the call. I'm going to use MR to load a digital model (sometimes called the digital twin) of an object and then add a conversational interface to allow the technician to learn more about how to service the equipment.

When you download the sample code from `bit.ly/2gF8LhP`, you'll get all the digital assets for this project. There are two separate Unity projects, one called Start and one called Complete. Start is an empty solution with only some digital assets to get you going. Complete contains all the scripts and digital assets for reference. I'll be working from Start but I'm also providing Complete for clarification of a few items.

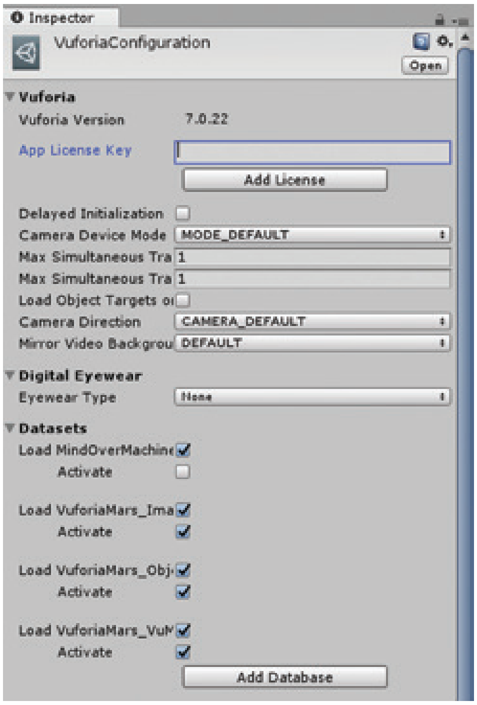
Building the First MR Scene

Target-based MR scenes—also known as marker-based augmented reality (AR)—have three components:

- ARCamera: This is the camera of the device viewing the MR scene.
- Target: This is the real-world object that triggers the digital content to appear.
- Stage: This is the world that the MR elements exist within.

When you open the Start project, you'll find only a Main Camera and Directional Light showing. Delete the Main Camera and click on Game Object | Vuforia | AR Camera. This will add the basic AR Camera to your scene. Click on the AR Camera and then on Open Vuforia Configuration. This is where you need to add the App License Key you created from the Vuforia Developer Portal, as shown in **Figure 1**. If you haven't done that already, click the Add License button to go through the tutorial. Unity 2017.3 is connected to the Vuforia developer workflow, which allows you to easily jump from Unity to the Vuforia Developer Portal.

Once your key is loaded, you can add targets. First, however, you need to have a target database for your MR app. Click the Add Database button (still in the Vuforia Configuration inspector). This opens the Vuforia portal where you can create a new target database—a collection of image or object targets that will trigger your MR content to appear. Create a new database, add some image targets and then download the Unity package of the database. Adding this package to your Unity project imports your image targets. One important note: image targets need to have a lot of tracking points to provide the best MR target scanning experience. I've included some sample targets in the Assets/images folder of the Start Unity project. Feel free to use these images for your image targets. Once you've downloaded and installed the Unity package for your target database, click on the checkbox (in Vuforia Configuration) by the name of the target database you just loaded. This loads the target database into Unity for your app to use. Finally, click the Activate checkbox to enable the target database to be available on app start.



VuforiaConfiguration

Vuforia
Vuforia Version 7.0.22
App License Key
Delayed Initialization ☐
Camera Device Mode MODE_DEFAULT
Max Simultaneous Tra 1
Max Simultaneous Tra 1
Load Object Targets o ☐
Camera Direction CAMERA_DEFAULT
Mirror Video Backgrou DEFAULT

Digital Eyewear
Eyewear Type None

Datasets
Load MindOverMachine ☑
Activate ☐
Load VuforiaMars_Ima ☑
Activate ☑
Load VuforiaMars_Obj ☑
Activate ☑
Load VuforiaMars_VuM ☑
Activate ☑

Figure 1 **The Vuforia Configuration Inspector**

> One important note: image targets need to have a lot of tracking points to provide the best MR target scanning experience.

Before you leave the Vuforia Configuration view, there's one more setting to check. Depending on your computer's camera, you might need to change the WebCam settings at the bottom of the screen. I'm developing on a Surface Pro 4, and its cameras aren't recognized by default. If you have a similar issue, open {Project Root}\Assets\Editor\QCAR\WebcamProfiles\profiles.xml and add the following XML:

```
<webcam deviceName="Microsoft Camera Rear">
  <windows>
    <requestedTextureWidth>640</requestedTextureWidth>
    <requestedTextureHeight>480</requestedTextureHeight>
    <resampledTextureWidth>640</resampledTextureWidth>
  </windows>
</webcam>
```

This will register the rear camera for the Surface Pro 4. If you want the front camera, just change the deviceName to Microsoft Camera Front.

Now the app is ready for an image target. Go to Game Object | Vuforia | Image. This will create an image target in your scene. In the Inspector | Image Target Behavior window, select Database to the target database you just loaded. Then select the image target you want to use from your database. The surface of the image target will now look like the image target selected.

To complete your MR scene, drag a 3D model from the Prefabs folder to be a child of the image target. When the AR Camera (which is the device's camera at run time) sees the image target, the child 3D model will appear in the world over the image target. At this point, your MR app is ready. Make sure you have a printed copy of the image target you selected and give the app a run.

Getting Conversational

Now that the app can see an image target and generate a digital object, let's create an interaction. I'm going to keep the interaction very simple and leverage a mix of the Windows Speech libraries that run on the device and the Speech API provided with Microsoft Cognitive Services. The questions the technician can ask, along with the answers, will be stored in the QnA Maker Service, one of the knowledge management offerings in Microsoft Cognitive Services. QnA Maker is a service that allows you to set up a question/answer pair using an existing FAQ, a formatted text file or even using the QnA input form through the service. Using your existing FAQ URLs or a text file exported from an existing knowledge management system lets you get up and running with the QnA Maker Service quickly and easily. Learn more about QnA Maker at qnamaker.ai.

I'm going to keep the interaction very simple and leverage a mix of the Windows Speech libraries that run on the device and the Speech API provided with Microsoft Cognitive Services.

Here's the flow: The user activates listening mode using a keyphrase; the user asks a question, which is compared against the QnA Maker Service to find an answer; the answer is returned as text, which is then sent to the Speech API for transformation into an audio file; the audio is returned and played to the user; the user deactivates listening mode using a keyphrase.

To keep this simple, I won't be using Language Understanding Intelligence Service (LUIS), but as you explore this solution, I encourage you to leverage LUIS to create a more natural conversation for users.

To start implementing the conversation components, I always create a Managers object to hold the various components needed to manage the scene overall. Create an empty Game Object called Managers and place it on the root of the scene.

Why Not Use HoloToolkit?

HoloToolkit is an excellent jumpstart to your HoloLens project. KeywordRecognizer already exists there and is much more robust than my sample, so why not use the HoloToolkit in this article? The reason is that I'm not building a HoloLens-only system. Instead, I'm setting the groundwork for you to build an app for the Surface Pro that can work on HoloLens but isn't constrained to it. Not every company is ready to take on the cost of a HoloLens for their fleet. The concepts presented here allow you to build for the Samsung GearVR, Google ARCore and even the Apple ARKit. If you're familiar with the HoloToolkit, you'll notice that I've borrowed some of the implementation because, as I said, it's very good. However, I've kept the code lighter and more concrete to work for the current scenario.

Users of this app will start the conversation by saying "Hello Central." This short statement is ideal for the KeywordRecognizer object. When the app hears Hello Central, it's akin to saying "Hey Cortana"—it triggers the app to listen for more commands.

First, I'll create the KeywordRecognizer to listen for the key words: Hello Central. To do this, I'll create a new C# script in the Scripts folder called KeywordRecognizer, then, within the class, I'll add a new class called KeywordAndResponse:

```
public class KeywordRecognizer : MonoBehaviour {
  [System.Serializable]
  public struct KeywordAndResponse {
    public string Keyword;
    public UnityEvent Response;
  }
  public KeywordAndResponse[] Keywords;
public bool StartAutomatically;
private UnityEngine.Windows.Speech.KeywordRecognizer recognizer;
```

Next, I'll implement the Start method to load the keywords and responses that will be provided through the Unity Editor interface to the app:

```
void Start () {
  List<string> keywords = new List<string>();
  foreach(var keyword in Keywords)
    keywords.Add(keyword.Keyword);

  this.recognizer =
    new UnityEngine.Windows.Speech.KeywordRecognizer(keywords.ToArray());
  this.recognizer.OnPhraseRecognized += KeywordRecognizer_OnPhraseRecognized;
  this.recognizer.Start();
}
```

In the Start method, I loop through each of the keywords to get a list of words. Then I register these words with the KeywordRecognizer, connect the OnPhraseRecognized event to a handler and finally start the recognizer. The OnPhraseRecognized method is called any time a phrase in the keywords list is heard by the app:

```
private void KeywordRecognizer_OnPhraseRecognized(
  PhraseRecognizedEventArgs args){
    foreach(var keyword in Keywords){
      if(keyword.Keyword == args.text){
        keyword.Response.Invoke();
        return;
    }
  }
}
```

This event loops through the word list to find the response, then calls the Invoke method to trigger the UnityEvent. By configuring the KeywordAndResponse object as I did, I can use the UnityEditor to configure the KeywordRecognizer. As stated before, this is a lite implementation of the the more reobust HoloToolkit KeywordRecognizer code.

Once KeywordRecognizer hears Hello Central, the app needs to listen for the next command. One challenge in building a knowledge management app like this is that knowledge can change. To make that change easy to capture and communicate across any platform, I'll use the QnA Maker Service to build a question-and-answer format for knowledge management. The QnA Maker Service is still in preview, but documentation can be found at qnamaker.ai.

I'm going to build another C# script component called QandAManager (create this script in your Scripts folder) to interact with the QnA Maker Service. Note that in this code I'll be using the UnityEngine.Windows.Speech namespace, which is available only in Windows. If you want to do this for Android or iOS,

Figure 2 **Stopping All KeywordRecognizers**

```
private void Recognizer_DictationResult(
  string text, ConfidenceLevel confidence) {
    if (confidence == ConfidenceLevel.Medium
      || confidence == ConfidenceLevel.High){
        currentQuestion = text;
    if (currentQuestion.ToLower() == EndCommandText.ToLower()) {
      StopListening();
    }
    else {
      GetResponse();
    }
  }
}
```

you need to implement the Speech to Text API from Microsoft Cognitive Services, which can be found at bit.ly/2hKYoJh.

I'll start the QandAManager with the following classes:

```
public class QandAManager : MonoBehaviour {
  [Serializable]
  public struct Question {
    public string question;
  }
  [Serializable]
  public struct Answer {
    public string answer;
    public double score;
  }
```

> Once KeywordRecognizer hears Hello Central, the app needs to listen for the next command.

These classes will be used to serialize the questions and answers to send to the QnA Maker Service. Next, I'll add some properties:

```
public string KnowledgeBaseId;
public string SubscriptionKey;
public AudioSource Listening;
public AudioSource ListeningEnd;
public string EndCommandText;
private DictationRecognizer recognizer;
private bool isListening;
private string currentQuestion;
```

KnowledgeBaseId will be used to determine which knowledge base to load, depending which image target is being viewed.

Figure 3 **The getResponse Coroutine**

```
private IEnumerator getResponse(){
  string url = "https://westus.api.cognitive.microsoft.com/" +
    "qnamaker/v1.0/knowledgebases/{0}/generateAnswer";
  url = string.Format(url, KnowledgeBaseId);
  Question q = new Question() { question = currentQuestion };
  string questionJson = JsonUtility.ToJson(q);
  byte[] questionBytes =
    System.Text.UTF8Encoding.UTF8.GetBytes(questionJson);

  Dictionary<string, string> headers = new Dictionary<string, string>();
  headers.Add("Ocp-Apim-Subscription-Key", SubscriptionKey);
  headers.Add("Content-Type", "application/json");

  WWW w = new WWW(url, questionBytes, headers);
  yield return w;
  if (w.isDone){
    Answer answer = JsonUtility.FromJson<Answer>(w.text);
    TextToSpeechManager tts = GetComponent<TextToSpeechManager>();
    tts.Say(answer.answer);
  }
}
```

SubsciptionKey is used to identify the app to the QnA Maker Service. Listening and ListeningEnd are audio cues to tell the user that the system is listening, or not. EndCommandText is the "hang up" command, in this case, "Thank You Central." The rest I'll discuss as they're implemented.

StartListening is the public method that triggers the QandAManager to start listening:

```
public void StartListening() {
  if (!isListening) {
    PhraseRecognitionSystem.Shutdown();
    recognizer = new DictationRecognizer();
    recognizer.DictationError += Recognizer_DictationError;
    recognizer.DictationComplete += Recognizer_DictationComplete;
    recognizer.DictationResult += Recognizer_DictationResult;
    recognizer.Start();
    isListening = true;
    Listening.Play(0);
  }
}
```

This turns on DictationRecognizer, connects the events and plays the sound to indicate the app is now listening. Only one recognizer can be running at a given time, which means KeywordRecognizer needs to be shut down before the DictationRecognizer can start. PhraseRecognitionSystem.Shutdown stops all KeywordRecognizers, as shown in **Figure 2**.

The Recognizer_DictationResult event takes the text the app hears and triggers the StopListening method if the EndCommandText phrase is recognized:

```
public void StopListening() {
  recognizer.Stop();
  isListening = false;
  ListeningEnd.Play(0);
  recognizer.DictationError -= Recognizer_DictationError;
  recognizer.DictationComplete -= Recognizer_DictationComplete;
  recognizer.DictationResult -= Recognizer_DictationResult;
  recognizer.Dispose();
  PhraseRecognitionSystem.Restart();
}
```

Otherwise, it performs GetResponse. For the sake of brevity, I won't go into the DictationError and DictationComplete events as they don't directly add to my solution. Please refer to the Complete Unity project to see an implementation.

StopListening shuts down the DictationRecognizer, disconnects all the events and then restarts the KeywordRecognizer. If the user initiates another interaction via Hello Central, the StartListening method will again be triggered and reconnect the DictationRecognizer.

The GetResponse method calls the getResponse coroutine:

```
private void GetResponse(){
  StartCoroutine(getResponse());
}
```

A coroutine is a function that can run across frames in Unity (for more information see bit.ly/2xJGT75). In this instance, getResponse is going to reach out to the QnA Maker Service, send a question and get back an answer, as shown in **Figure 3**. If you're not a Unity developer, the JSONUtility class might be new to you. JSONUtility is a class built into Unity to serialize JSON objects to and from C# objects. In **Figure 3**, I'm using JsonUtility.ToJson to convert the Question object to a JSON representation to send to the QnA Maker Service.

For brevity I didn't include checking w.error to handle any error that comes from the QnA Maker Service, but you should make sure to handle errors accordingly.

In the code in **Figure 3**, you can see a reference to the TextToSpeechManager, which is the next component I'll build. As its name suggests, TextToSpeechManager will take the text and turn it into an audio file. Before I do this, though, I'll first build a component that allows the app to determine which knowledge base ID to use based on which image target is being viewed.

Identifying the Target

In Vuforia, targets can implement the ITrackableEventHandler to perform custom actions when the target's recognition status changes or even when the user changes from one target to another. In this app, I'm going to update the QandAManager KnowledgeBaseId property when the target is recognized.

> One challenge in building a knowledge management app like this is that knowledge can change.

I'll create a new C# script in the Scripts folder called TargetTracker and add it to the Managers game object. Inside TargetTracker, I'll add the ITrackableEventHandler interface:

```
public class TargetTracker : MonoBehaviour, ITrackableEventHandler {
  private TrackableBehaviour mTrackableBehaviour;
  public QandAManager qnaManager;
  public string KnowledgeBaseId;
```

The public KnowledgeBaseId value is a string you set to identify which KnowledgeBaseId to use when the target is recognized. The qnaManager object is a reference to the QandAManager component. Now I'll configure the component in Start to connect the TrackableBehavior and QandAManger:

```
void Start () {
  mTrackableBehaviour = GetComponent<TrackableBehaviour>();
  if (mTrackableBehaviour) {
    mTrackableBehaviour.RegisterTrackableEventHandler(this);
  }
}
```

Finally, I'll set the OnTrackableStateChanged event to update the knowledge base Id of the QandAManager with the knowledge base Id for this component:

```
public void OnTrackableStateChanged(
  TrackableBehaviour.Status previousStatus,
  TrackableBehaviour.Status newStatus){
    if (newStatus == TrackableBehaviour.Status.DETECTED ||
      newStatus == TrackableBehaviour.Status.TRACKED ||
      newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED){
        qnaManager.KnowledgeBaseId = KnowledgeBaseId;
  }
}
```

With the code complete you can add the TargetTracker component to the Image Target in your scene. Next, take the subscription key from the QnA Maker Service and enter it into the QandAManger subscription key. Then take the knowledge base Id provided by the QnA Maker Service and add that to the TargetTracker associated with your scene's image target. If you have multiple knowledge bases, you just need to update the knowledge base Id per target to switch which questions and answers users can ask per target.

When setting up the QnA Maker Service, remember to format your questions the way you expect users to ask them. Developers with experience working with users know that predicting user behavior, especially how users will ask a question, is very difficult. To help you build better questions and answers, consider implementing a telemetry tool like Application Insights or another logging mechanism to capture the questions users are asking. This will help you tailor your UX to deliver the best answers to the most common questions.

Talking Back

A conversation goes two ways. One way is the user talking to the app; the other is the app talking to the user. HoloToolkit has a great text-to-speech control, but for this article I'm going to set up and use the Cognitive Services Text to Speech API. Using the Cognitive Services Text to Speech capability allows the app to be more portable across platforms.

Before I start coding the solution, I should note that the Unity classes WWW, WWWForm and UnityWebRequest have a few limitations that create challenges for the code I'm about to write. One such limitation is that to get a token in the Speech API, you must POST an empty body to the token URL. WWW, WWWForm and UnityWebRequest all require you to provide something in the body of a post. But if you provide anything in the body, the Speech API will return a 400 Bad Request error. To solve this, I'm going to be using the C# WebRequest class, which doesn't support coroutines. To this end, I'll get the token for the speech API during the Start method of the component.

I'll create a new C# script in the Scripts folder called TextToSpeechManager and add it to the Managers game object. This component will hold all of my text-to-speech capabilities and expose a public method called Say(string text) to be used by the QnAManager for saying the answer returned from the QnA Service. I'll start the component with a few public variables:

```
public class TextToSpeechManager : MonoBehaviour {
  public string SubscriptionKey;
  public string SSMLMarkup;
  public Genders Gender = Genders.Female;
  public AudioSource audioSource;
  private string token;
  private string ssml;
```

SubscriptionKey stores the token the app will need to get an authentication token from the Speech API. This key is obtained from the Azure Portal by adding the Cognitive Services Speech API. SSMLMarkup is the XML content I'll use to generate speech from the Speech API. For this article I've set up the default value to be:

```
<speak version='1.0' xml:lang='en-US'>
  <voice xml:lang='en-US'
    xml:gender='{0}'
    name='Microsoft Server Speech Text to Speech Voice (en-US, ZiraRUS)'>{1}</voice>
</speak>
```

Here, {0} will be replaced with the Gender value and {1} will be the text to send to the Speech API.

Now I'll add the Start method:

```
void Start(){
  ServicePointManager.ServerCertificateValidationCallback =
    remoteCertificateValidationCallback;
  token = getToken();
}
```

Start begins by setting up a custom server certificate validation so that Unity can handle the authentication process with the Speech

API. The complete code for the validation can be found in the Complete Unity project. Next, I call getToken, which connects to the Speech API authentication service and returns the token to use for future calls to the API:

```
private string getToken(){
  WebRequest request =
    WebRequest.Create("https://api.cognitive.microsoft.com/sts/v1.0/issueToken");
  request.Headers.Add("Ocp-Apim-Subscription-Key",
    SubscriptionKey);
  request.Method = "POST";
  WebResponse response = request.GetResponse();
  Stream responseStream = response.GetResponseStream();
  StreamReader reader = new StreamReader(responseStream);
  return reader.ReadToEnd();
}
```

The getToken method is a straightforward connection to the issueToken endpoint, passing in the subscription key and reading the token as a string returned from the API service. However, the token is obtained at start and might time out. In the Complete Unity project, you'll find token refresh code.

> The goal of combining MR and Cognitive Services is to create immersive experiences in which the physical and digital worlds blur into one cohesive interaction.

The Say method is the public method called in the QandAManger. The signature for this method has a few different parts, with the basic flow being: prepare the SSML, send it to the Speech API, get the audio file, save the file and then load the file as an audio clip.

```
public void Say(string text){
  ssml = string.Format(
    SSMLMarkup,
    Enum.GetName(typeof(Genders), Gender),
    text);
  byte[] ssmlBytes =
    System.Text.UTF8Encoding.UTF8.GetBytes(ssml);
```

This first block of code gets the ssml and prepares it for transmission to the API by converting it to a byte array. Next, I prepare the request to the Speech API:

```
HttpWebRequest request =
  (HttpWebRequest)WebRequest.Create("https://speech.platform.bing.com/synthesize");
request.Method = "POST";
request.Headers.Add("X-Microsoft-OutputFormat", "riff-16khz-16bit-mono-pcm");
request.Headers.Add("Authorization", "Bearer " + token);
request.ContentType = "application/ssml+xml";
request.SendChunked = false;
request.ContentLength = ssmlBytes.Length;
request.UserAgent = "ContosoEnergy";
```

In the headers I add X-Microsoft-OutputFormat and set it to riff-16khz-16bit-mono-pcm so the Speech API will return a .wav file. Unity needs a .wav file to stream the audio clip later in the say method. Notice the request.SendChunked = false statement. This ensures the Transfer Encoding property isn't set to chunked, which would cause a timeout (408) error when connecting to the Speech API. I also update the UserAgent to be ContosoEnergy because the default Unity value is too long for the Speech API to accept.

With the request and headers prepared, I write the ssmlBytes to the request stream:

```
Stream postData = request.GetRequestStream();
postData.Write(ssmlBytes, 0, ssmlBytes.Length);
postData.Close();
```

Next, I get the response and prepare the file path for saving the audio file that came back from the Speech API:

```
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
string path = string.Format("{0}\\assets\\tmp\\{1}.wav",
  System.IO.Directory.GetCurrentDirectory(),
  DateTime.Now.ToString("yyyy_mm_dd_HH_nn_ss"));
```

Then I save the audio file and call the say coroutine:

```
using (Stream fs = File.OpenWrite(path))
using (Stream responseStream = response.GetResponseStream()){
  byte[] buffer = new byte[8192];
  int bytesRead;
  while ((bytesRead =
    responseStream.Read(buffer, 0, buffer.Length)) > 0){
      fs.Write(buffer, 0, bytesRead);
  }
}
StartCoroutine(say(path));
}
```

The last method needed here is the coroutine to read the .wav file and play it as audio. The WWW object has a built-in www.GetAudioClip function that makes it easy to load and play a file:

```
private IEnumerator say(string path) {
  WWW w = new WWW(path);
  yield return w;
  if (w.isDone) {
    audioSource.clip = w.GetAudioClip(false, true, AudioType.WAV);
    audioSource.Play();
  }
}
```

As before, I omitted the w.error check for brevity, but always make sure to handle errors in your code.

Wrapping Up

In this article I built a basic marker-based MR application using Vuforia and Unity. By using image targets, I set up the basic capability to scan an image to generate digital content. I then implemented the Windows KeywordRecognizer and DictationRecognizer components to allow the app to listen to the user. I took the text from the DictationRecognizer and enabled the app to respond to questions via the QnA Maker Service. Finally, I enabled the app to talk back to the user with the Cognitive Services Speech API.

The goal of combining MR and Cognitive Services is to create immersive experiences in which the physical and digital worlds blur into one cohesive interaction. I encourage you to download the code and extend it with LUIS or the Computer Vision APIs to bring the user even further into your app. ■

Tim Kulp *is the director of Emerging Technology at Mind Over Machines in Baltimore, Md. He's a mixed reality, artificial intelligence and cloud app developer, as well as author, painter, dad, and "wannabe mad scientist maker." Find him on Twitter: @tim_kulp or via LinkedIn: linkedin.com/in/timkulp.*

Thanks to the following technical experts for reviewing this article: *Alessandro Del Sole (Microsoft) and Will Gee (BaltiVirtual)*

Visual C++ Support for Stack-Based Buffer Protection

Hadi Brais

When software does something it's not supposed to do according to its functional specification, it's said to have defects or bugs. The rules in that specification that determine when accesses and modifications to data and other resources should be allowed collectively constitute a security policy. The security policy essentially defines what it means for the software to be secure, and when a particular defect should be considered as a security flaw rather than just another bug.

Given various threats from around the world, security is more important than ever today and, as such, must be an integral part of the software development lifecycle (SDL). This includes choices such as where to store data, which C/C++ runtime APIs to use, and which tools can help make the software more secure. Following the C++ Core Guidelines (`bit.ly/1LoeSRB`) substantially helps in writing correct, maintainable code. In addition, the Visual C++ compiler offers many security features that are easily accessible through compiler switches. These can be classified as either static or dynamic security analyses. Examples of static security checks include using the /Wall and /analyze switches and the C++ Core Guidelines checkers. These checks are performed statically and don't affect the generated code, though they do increase compilation time. In contrast, dynamic checks are inserted in the emitted executable binaries by the compiler or the linker. I'll discuss in this article specifically one dynamic security analysis option, namely /GS, which provides protection against stack-based buffer overflows. I'll explain how the code is transformed when that switch is turned on and when it can or can't secure your code. I'll be using Visual Studio Community 2017.

You might wonder, why not just turn on all these compiler switches and be done with it. In general, you should employ all the recommended switches regardless of whether you understand how they work. However, knowing the details of how a particular technique works enables you to determine the impact that it may have on your code and how to better make use of it. Consider, for example, buffer overflows. The compiler does offer a switch to deal with such defects, but it uses a detection mechanism that forces the program to crash when a buffer overflow is detected. Does that improve security? It depends. First, while all buffer overflows are bad, not all are security vulnerabilities and so it doesn't necessarily mean an exploitation took place. And even if it did, the damage might have already been done by the time the detection mechanism was triggered. Moreover, depending on how your application is designed, abruptly crashing the program may not be suitable

This article discusses:

- Control flow attacks
- GuardStack and the /GS switch
- Using BinSkim to verify GuardStack

Technologies discussed:

Visual C++, GuardStack, BinSkim

because it could by itself be a denial-of-service (DoS) vulnerability or lead to a potentially worse situation involving data loss or corruption. As I'll explain in this article, the only reasonable thing to do is to make the application resilient to such crashes, rather than disabling or changing the protection mechanism.

I've written a number of articles on compiler optimizations for *MDSN Magazine* (you'll find the first at `msdn.com/magazine/dn904673`). The goal was mainly to improve execution time. Security can also be viewed as an objective of compiler transformations. That is, rather than optimizing execution time, security would be optimized by reducing the number of potential security flaws. This perspective is useful because it suggests that when you specify multiple compiler switches to improve both execution time and security, the compiler might have multiple, potentially conflicting goals. In this case, it has to somehow balance or prioritize these goals. I'll discuss the impact that /GS has on some aspects of your code, particularly speed, memory consumption, and executable file size. That's another reason to understand what these switches do to your code.

In the next section, I'll provide an introduction to control flow attacks with particular focus on stack buffer overflows. I'll discuss how they occur and how an attacker can exploit them. Then I'll look in detail at how /GS impacts your code and the degree to which it can mitigate such exploits. Finally, I'll demonstrate how to use the BinSkim static binary analysis tool to perform a number of critical verification checks on a given executable binary without requiring the source code.

Control Flow Attacks

A buffer is a block of memory used to temporarily store data to be processed. Buffers can be allocated from the runtime heap, the thread stack, directly using the Windows VirtualAlloc API, or as a global variable. Buffers can be allocated from the runtime heap either using the C memory allocation functions (such as malloc) or the C++ new operator. Buffers can be allocated from the stack using either an automatic array variable or the _alloca function. The minimum size of a buffer can be zero bytes and the maximum size depends on the size of the largest free block.

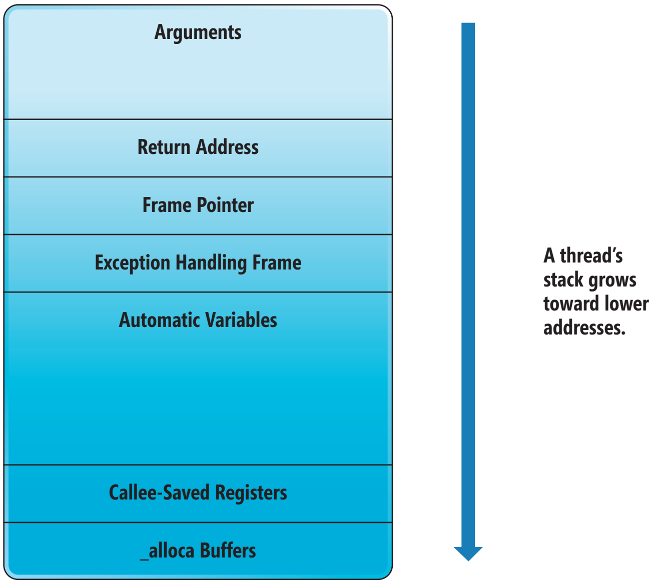


Figure 1 **A Typical x86 Stack Frame**

Two particular features of the C and C++ programming languages that truly distinguish them from other languages, such as C#, are:

- You can do arbitrary arithmetic on pointers.
- You can successfully dereference any pointer anytime as long as it points to allocated memory (from the point of view of the OS), though the behavior of the application may be undefined if it doesn't point at the memory it owns.

> The minimum size of a buffer can be zero bytes and the maximum size depends on the size of the largest free block.

These features make the languages very powerful, but they constitute a great threat at the same time. In particular, a pointer that's intended to be used to access or iterate over contents of a buffer might be erroneously or maliciously modified so that it points outside the bounds of the buffer to either read or write adjacent or other memory locations. Writing beyond the largest address of a buffer is called a buffer overflow. Writing before the smallest address of a buffer (which is the address of the buffer) is called a buffer underflow.

A stack-based buffer overflow vulnerability has been discovered recently in an extremely popular piece of software (which I won't name). This resulted from using the sprintf function unsafely, as shown in the following code:

```
sprintf(buffer, "A long format string %d, %d", var1, var2);
```

The buffer is allocated from the thread stack and it's a fixed size. However, the size of the string to be written to the buffer depends on the number of characters required to represent two specified integers. The size of the buffer isn't sufficient to hold the largest possible string, resulting in a buffer overflow when large integers are specified. When an overflow occurs, adjacent memory locations higher up in the stack get corrupted.

To demonstrate why this is dangerous, consider where a buffer allocated from the stack would typically be located in the stack frame of the declaring function according to standard x86 calling conventions and taking into consideration compiler optimizations, as shown in **Figure 1**.

First, the caller pushes any arguments that aren't passed through registers onto the stack in a certain order. Then, the x86 CALL instruction pushes the return address onto the stack and jumps to the first instruction in the callee. If frame pointer omission (FPO) optimization doesn't take place, the callee pushes the current frame pointer onto the stack. If the callee uses any exception-handling constructs that haven't been optimized away, an exception-handling frame would next be placed onto the stack. That frame contains pointers to and other information about exception handlers defined in the callee. Non-static local variables that haven't been optimized away and that can't be held in registers or that spill from registers are allocated from the stack in a certain order. Next, any callee-saved registers used by the callee must be saved on the stack. Finally, dynamically sized buffers allocated using _alloca are placed at the bottom of the stack frame.

Any of the data items on the stack may have certain alignment requirements, so padding blocks may be allocated as required. The piece of code in the callee that sets up the stack frame (except for the arguments) is called the prolog. When the function is about to return to its caller, a piece of code called the epilog is responsible for deallocating the stack frame up to and including the return address.

The main difference between the x86 and ARM calling conventions is that the return address and frame pointer are held in dedicated registers in ARM rather than on the stack.

The main difference between the x86/x64 and ARM calling conventions is that the return address and frame pointer are held in dedicated registers in ARM rather than on the stack. Nonetheless, stack buffer out-of-bounds accesses do constitute a serious security issue on ARM because other values on the stack may be pointers.

A stack buffer overflow (writing beyond the upper bound of a buffer) may overwrite any of the code or data pointers that are stored above the buffer. A stack buffer underflow (writing below the lower bound of a buffer) may overwrite the values of the callee-saved registers, which may also be code or data pointers. An arbitrary out-of-bounds write will either cause the application to crash or behave in an undefined way. However, a maliciously crafted attack enables the attacker to take control of the execution of the application or the whole system. This can be achieved by overwriting a code pointer (such as the return address) so that it points to a piece of code that executes the attacker's intent.

GuardStack (GS)

To mitigate stack-based out-of-bounds accesses, you could manually add the necessary bounds checks (adding if statements to check that a given pointer is within the bounds) or use an API that performs these checks (for example, snprintf). However, the vulnerability may still persist for different reasons, such as incorrect integer arithmetic or type conversions used to determine the bounds of buffers or to perform bounds checking. Therefore, a dynamic mitigation mechanism to prevent or reduce the possibility of exploitation is required.

General mitigation techniques include randomizing the address space and using non-executable stacks. Dedicated mitigation techniques can be classified according to whether the goal is to prevent out-of-bounds accesses from occurring by capturing them before they occur, or to detect out-of-bounds accesses at some point after they occur. Both are possible, but prevention adds substantial performance overhead.

The Visual C++ compiler offers two detection mechanisms that are somewhat similar, but have different purposes and different performance costs. The first mechanism is part of the runtime error checks, which can be enabled using the /RTCs switch. The second is GuardStack (called Buffer Security Check in the documentation and Security Check in Visual Studio), which can be enabled using the /GS switch.

With /RTCs, the compiler allocates additional small memory blocks from the stack in an interleaving manner such that every local variable on the stack is sandwiched between two such blocks. Each of these additional blocks is filled with a special value (currently, 0xCC). This is handled by the prolog of the callee. In the epilog, a runtime function is called to check whether any of these blocks were corrupted and report a potential buffer overflow or underflow. This detection mechanism adds some overhead in terms of performance and stack space, but it's designed to be used for debugging and ensuring program correctness, not just as a mitigation.

GuardStack, on the other hand, was designed to have lower overhead and as a mitigation that can actually work in a production, potentially malicious, environment. So /RTCs should be used for debug builds and GuardStack should be used for both builds. In addition, the compiler doesn't allow you to use /RTCs with compiler optimizations, while GuardStack is compatible and doesn't interfere with compiler optimizations. By default, both are enabled in the Debug configuration while only GuardStack is enabled in the Release configuration of a Visual C++ project. In this article, I'll only discuss GuardStack in detail.

When GuardStack is enabled, a typical x86 call stack would look like what's shown in **Figure 2**.

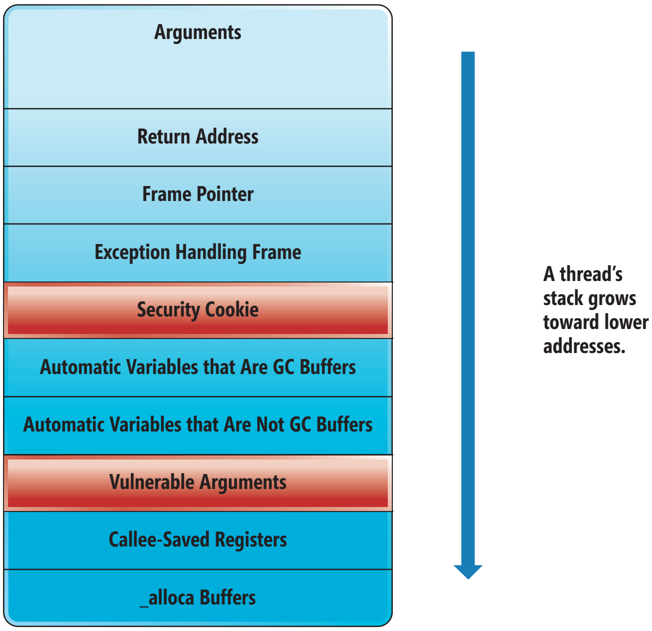


Figure 2 **A Typical x86 Stack Frame Protected Using GuardStack (/GS)**

There are three differences compared to the stack layout shown in **Figure 1**. First, a special value, called a cookie or a canary, is allocated just above the local variables. Second, local variables that are more likely to exhibit overflows are allocated above all other local variables. Third, some of the arguments that are particularly sensitive to buffer overflows are copied to an area below local variables. Of course, to make these changes happen, a different prolog and epilog are used, as I'll discuss now.

The prolog of a protected function would include roughly the following additional instructions on x64:

```
sub       rsp,8h
mov       rax,qword ptr [__security_cookie]
xor       rax,rbp
mov       qword ptr [rbp],rax
```

An additional 8 bytes is allocated from stack and is initialized to a copy of the value of the __security_cookie global variable XOR'd with the value held in the RBP register. When /GS is specified, the compiler automatically links the object file built from gs_cookie.c source file. This file defines __security_cookie as a 64-bit or 32-bit global variable of the type uintptr_t on x64 and x86, respectively. Therefore, each Portable Executable (PE) image compiled with /GS includes a single definition of that variable used by the prologs and epilogs of the functions of that image. On x86, the code is the same except that 32-bit registers and cookies are used.

The basic idea behind using a security cookie is to detect, just before the function returns, whether the value of the cookie has become different from that of the reference cookie (the global variable). This indicates a potential buffer overflow caused by either an exploitation attempt or just an innocent bug. It's crucial that the cookie has very high entropy to make it extremely difficult for an attacker to guess. If an attacker is able to determine the cookie used in a particular stack frame, GuardStack fails. I'll discuss more about what GuardStack can and can't do later in this section.

It's crucial that the cookie has very high entropy to make it extremely difficult for an attacker to guess.

The reference cookie is given an arbitrary constant value when the image is emitted by the compiler. Therefore, it must be carefully initialized, basically before any code is executed. Recent versions of Windows are aware of GuardStack and will initialize the cookie to a high-entropy value at load time. When /GS is enabled, the first thing the entry point of an EXE or DLL does is initialize the cookie by calling the __security_init_cookie defined in gs_support.c and declared in process.h. This function initializes the image's cookie if it hasn't been appropriately initialized by the Windows loader.

Note that without XOR'ing with RBP, merely leaking the reference cookie at any point during execution (using an out-of-bounds read, for example) is sufficient to subvert GuardStack. XOR'ing with RBP enables you to efficiently generate different cookies and the attacker would need to know both the reference cookie and the RBP to figure out the cookie for one stack frame. RBP by itself isn't guaranteed to have high entropy because its value depends on how the compiler optimized the code, the stack space consumed so far, and the randomization performed by address space layout randomization (ASLR), if enabled.

To minimize overhead, only those functions that the compiler considers vulnerable are protected.

The epilog of a protected function would include roughly the following additional instructions on x64:

```
mov       rcx,qword ptr [rbp]
xor       rcx,rbp
call      __security_check_cookie
add       esp,8h
```

First, the cookie on the stack is XOR'd to produce a value that's supposed to be the same as the reference cookie. The compiler emits instructions to ensure that the value of RBP used in the prolog and epilog is the same (unless it somehow got corrupted).

The __security_check_cookie function, declared in vcruntime.h, is linked by the compiler and its purpose is to validate the cookie that's on the stack. This is done mainly by comparing the cookie with the reference cookie. If the check fails, the code jumps to the __report_gsfailure function, which is defined in gs_report.c. On Windows 8 and later, the function terminates the process by calling __fastfail. On other systems, the function terminates the process by calling UnhandledExceptionFilter after removing any potential handler. Either way, the error is logged by Windows Error Reporting (WER) and it contains information in which stack frame the security cookie got corrupted.

When /GS was first introduced in Visual C++ 2002, you could override the behavior of a failed stack cookie check by specifying a callback function. However, since the stack is in an undefined state and since some code already got executed before the overflow was detected, there's almost nothing that can be reliably done at that point. Therefore, later versions starting with Visual C++ 2005 eliminated this feature.

The Overhead of GuardStack

To minimize overhead, only those functions that the compiler considers vulnerable are protected. Different versions of the compiler may use different undocumented algorithms to determine whether a function is vulnerable, but in general, if a function defines an array or a large data structure and obtains pointers to such objects, it's likely that it will be considered vulnerable. You can specify that a particular function not be protected by applying __declspec(safebuffers) to its declaration. However, this keyword is ignored when applied to a function that's inlined in a protected function or when a protected function is inlined in it. You can also force the compiler to protect one or more functions using the strict_gs_check pragma. The security development lifecycle (SDL) checks, enabled using /sdl, specifies strict GuardStack on all source files and other dynamic security checks.

GuardStack copies vulnerable parameters to a safer location below local variables so that if an overflow occurrs, it would be more difficult to corrupt those parameters. A parameter that's a pointer or a C++ reference may qualify as a vulnerable parameter. Refer to the documentation on /GS for more information.

I've conducted a number of experiments using C/C++ production applications to determine the overhead related to both performance and image size. I've applied strict_gs_check on all source files so the results are independent of what the compiler considers vulnerable functions (I refrained from using /sdl because it enables other security checks, which have their own overheads). The largest performance overhead I got was 1.4 percent and the largest image size overhead was 0.4 percent. The worst-case scenario would occur in a program that spends most of its time calling protected functions that do very little work. Well-designed real programs don't exhibit such behavior. Keep in mind also that GuardStack incurs a potentially non-negligible stack space overhead.

On the Effectiveness of GuardStack

GuardStack is designed to mitigate only a specific type of vulnerability, namely stack buffer overflow. More important, using GuardStack by itself against this vulnerability may not provide a high degree of protection because there are ways for an attacker to go around it:

- The detection of a corrupt cookie occurs only when the function returns. A lot of code might get executed between the time the cookie is corrupted and the time that corruption is detected. That code might be using other values from the stack, above or below the cookie, that have been overwritten. This creates an opportunity for an attacker to take (partial) control of the execution of the application. In that case, detection may not even take place at all.

GuardStack is designed to mitigate only a specific type of vulnerability, namely stack buffer overflow.

- A buffer overflow can still occur without overwriting the cookie. The most dangerous case would be overflowing a buffer allocated using _alloca. Even protected arguments and callee-saved registers can be overwritten in this case.
- It may be possible to leak some of the cookies using out-of-bounds memory reads. Because different images use different reference cookies, and because cookies are XOR'd with the RBP, it can be more challenging for an attacker to make use of leaked cookies. However, the Windows Subsystem for Linux (WSL) might have introduced another way to leak cookies. WSL provides an emulation of the fork Linux system call by which a new process is created that duplicates the parent process. If the application being attacked forks a new process to handle incoming client requests, a malicious client can issue a fairly small number of requests to determine the values of the security cookies.
- A number of techniques have been proposed to guess an image's reference cookie in certain situations. I'm not aware of any successful real attacks in which the reference cookie was guessed, but the probability of success isn't tiny enough to dismiss it. XOR'ing with RBP adds another very important layer of defense against such attacks.

Combining a custom entry point with the effects of /GS can lead to interesting scenarios.

- GuardStack mitigates vulnerabilities by introducing different potential vulnerabilities, in particular, DoS and data loss. When the corruption of a cookie is detected, the application is abruptly terminated. For a server application, the attacker can cause the server to crash, potentially losing or corrupting valuable data.

Therefore, it's important that you first strive to write correct, secure code with the help of static analysis tools. Then, following the defense-in-depth strategy, employ GuardStack and other dynamic mitigations offered by Visual C++ (many of which are enabled by default in the Release build) in the code you ship.

/GS with /ENTRY

The default entry point function (*CRTStartup) specified by the compiler when you compile to produce an EXE or a DLL file does four things in order: initializes the reference security cookie; initializes the C/C++ runtime; calls the main function of your application; and terminates the application. You can use the /ENTRY linker switch to specify a custom entry point. However, combining a custom entry point with the effects of /GS can lead to interesting scenarios.

The custom entry point and any functions it calls are candidates for protection. If the Windows loader appropriately initialized the cookie, then any protected functions will use a copy of a reference cookie that's the same in their prologs and epilogs. So no problem will occur.

If Windows didn't appropriately initialize the cookie and the first thing the custom entry point does is to call __security_init_cookie, then all protected functions will use the correct reference cookie except for the entry point. Recall that a copy of the reference cookie is made in the epilog. Therefore, if the entry point returns normally, the cookie will be checked in its epilog and the check will fail, resulting in a false positive. To avoid this problem, you should call a function to terminate the program (such as exit) rather than returning normally.

If Windows didn't appropriately initialize the cookie and the entry point didn't call __security_init_cookie, then all protected functions will use the default reference cookie. Fortunately, since this cookie is XOR'd with RBP, the entropies of the used cookies won't be zero. So you'll still get some protection, especially with ASLR. However, it's recommend that you properly initialize the reference cookie by calling __security_init_cookie.

Using BinSkim to Verify GuardStack

BinSkim is a light, static binary analysis tool that verifies the correctness of the usage of some of the security features used in a given PE binary. One particular feature that BinSkim supports is GuardStack. BinSkim is open source (github.com/Microsoft/binskim) under MIT license and written completely in C#. It supports x86, x64 and ARM Windows binaries that are compiled with recent versions of Visual C++ (2013+). You can either use it as a stand-alone tool or, more interestingly, include (part of) it in your code. For example, if you have an application that supports PE plug-ins, you can use BinSkim to verify that a plug-in employs the recommended security features and refuse to load it otherwise. I'll discuss in this section how to use BinSkim as a stand-alone tool.

As far as GuardStack is concerned, the tool verifies that the specified binary adheres to the following four rules:

- EnableStackProtection: Checks the corresponding flag that's stored in the associated PDB file. If the flag isn't found, the rule fails. Otherwise, it passes.
- InitializeStackProtection: Iterates the list of global functions as defined in the associated PDB file to find the functions __security_init_cookie and __security_check_cookie. If both aren't found, the tool considers that /GS wasn't enabled. In this case, EnableStackProtection should fail. If __security_init_cookie wasn't defined, the rule fails. Otherwise, it passes.
- DoNotModifyStackProtectionCookie: Looks up the location of the reference cookie using the load configuration data of the image. If the location isn't found, the rule fails. If the load configuration data indicates that a cookie is defined, but its offset is invalid, the rule fails. Otherwise, the rule passes.

> The GuardStack dynamic mitigation technique is an extremely important detection-based mitigation against stack buffer overflow vulnerabilities.

- DoNotDisableStackProtectionForFunctions: Uses the associated PDB file to determine if there are any functions with the __declspec(safebuffers) attribute applied on them. The rule fails if any are found. Otherwise, it passes. Using __declspec(safebuffers) is disallowed by the Microsoft SDL.

To use BinSkim, first download the source code from the GitHub repository and build it. To run BinSkim, execute the following command in your favorite shell:

```
binskim.exe analyze target.exe --output results.sarif
```

Figure 3 **BinSkim Analysis Results File**

```
{
  "ruleId": "BA2014",
  "level": "pass",
  "formattedRuleMessage": {
    "formatId": "Pass ",
    "arguments": [
      "myapp.exe",
    ]
  },
  "locations": [
    {
      "analysisTarget": {
        "uri": "file:///D:/src/build/myapp.exe"
      }
    }
  ]
}
```

To analyze more than one image, you can use the following command:

```
binskim.exe analyze myapp\*.dll --recurse --output results.sarif --verbose
```

Note that you can use wild cards in file paths. The --recurse switch specifies that BinSkim should analyze images in subdirectories, too. The --verbose switch tells BinSkim to include in the results file the rules that passed—not just the ones that failed.

The results file is in the Static Analysis Results Interchange Format (SARIF). If you open it in a text editor, you'll find entries that look like what's shown in **Figure 3**.

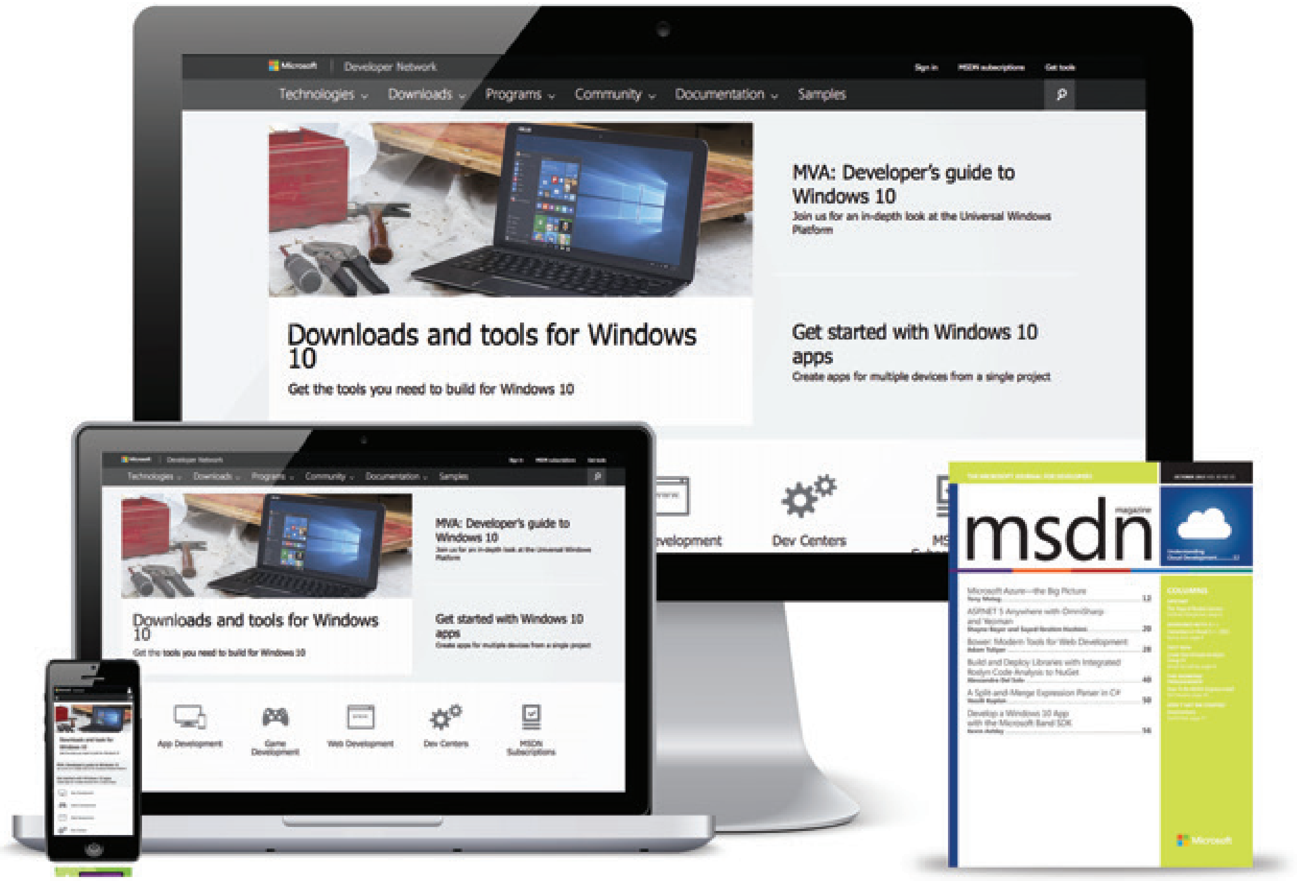
Every rule has a rule ID. The rule ID BA2014 is the ID of the DoNotDisableStackProtectionForFunctions rule. The Microsoft SARIF SDK (github.com/Microsoft/sarif-sdk) includes the source code of a Visual Studio extension that views SARIF files in Visual Studio.

Wrapping Up

The GuardStack dynamic mitigation technique is an extremely important detection-based mitigation against stack buffer overflow vulnerabilities. It's enabled by default in both the Debug and Release builds in Visual Studio. It was designed to have a negligible overhead for most programs so that it can be widely used. However, it doesn't provide an ultimate solution for such vulnerabilities. Buffer overflows are common for buffers allocated from the stack, but they can also occur in any allocated memory region. Most prominently, heap-based buffer overflows are just as perilous. For these reasons, it's very important to use other mitigation techniques offered by Visual C++ and Windows such as Control Flow Guard (CFG), Address Space Layout Randomization (ASLR), Data Execution Prevention (DEP), Safe Structured Exception Handling (SAFESEH), and Structured Exception Handling Overwrite Protection (SEHOP). All of these techniques work synergistically to harden your application. For more information on these techniques and others, refer to bit.ly/2iLG9rq. ■

HADI BRAIS *is a doctorate scholar at the Indian Institute of Technology Delhi, researching compiler optimizations, computer architecture, and related tools and technologies. He blogs on hadibrais.wordpress.com and can be contacted at hadi.b@live.com.*

THANKS to the following technical experts for reviewing this article: *Shayne Hiet-Block (Microsoft), Mateusz Jurczyk (Google), Preeti Ranjan Panda (IITD), Andrew Pardoe (Microsoft)*


Microsoft | Developer Network
Technologies
Downloads
Programs
Community
Documentation
Samples
MVA: Developer's guide to Windows 10
Join us for an in-depth look at the Universal Windows Platform
Downloads and tools for Windows 10
Get the tools you need to build for Windows 10
Get started with Windows 10 apps
Create apps for multiple devices from a single project
App Development
Game Development
Web Development
Dev Centers
MSDN Subscriptions
msdn magazine

CUTTING EDGE

DINO ESPOSITO

Configuration of ASP.NET Core Applications

The logic of any realistic software application depends on some external configuration data that, when fetched, drives the overall behavior of the application. Generally speaking, there are three types of configuration data: data fetched once and used everywhere, data fetched frequently and used everywhere, and data fetched on-demand just before use. The implementation of the latter two types of configuration data are, for the most part, application-specific. The first type, data fetched only once in the application's lifetime, tends to address the building of a wrapper API that hides the actual data store as much as possible.

In classic ASP.NET, the web.config file was, among many other things, the favorite repository of application-wide configuration data. In ASP.NET Core, there's no web.config file anymore, yet the configuration API is richer and more flexible than ever before.

Configuration Data Providers

The configuration API is centered on the concept of the data provider. The data provider retrieves data from a given source and exposes it to the application in the form of name/value pairs. ASP.NET Core comes with a few predefined configuration providers able to read from text files (most notably JSON files), environment variables and in-memory dictionaries. Configuration providers are plugged into the system at application startup through the services of the ConfigurationBuilder class. All providers linked to the builder contribute their own name/value pairs and the resulting collection is exposed as an IConfigurationRoot object.

While data always comes in as a name/value pair, the overall structure of the resulting configuration object might be hierarchical, as well. It all depends on the actual nature and structure of the value associated with the name. Taken from the constructor of a startup class, the following code snippet shows how to build a configuration tree combining together two different configuration providers:

```
// Env is a reference to the IHostingEnvironment instance
// that you might want to inject in the class via ctor
var dom = new ConfigurationBuilder()
  .SetBasePath(env.ContentRootPath)
  .AddJsonFile("MyAppSettings.json")
  .AddInMemoryCollection(new Dictionary<string, string> {{"Timezone", "+1"}})
  .Build();
```

The AddJsonFile extension method adds name/value pairs from the properties stored in the specified JSON file. Note that the JSON file is listed through a relative path. The SetBasePath method, in fact, sets the root directory where the system will start looking for any such referenced files. Any JSON files can be used as a configuration data source. The structure of the files is completely up to you and can include any level of nesting. Multiple JSON files can be linked at the same time.

The AddInMemoryCollection method adds the content of the specified dictionary. Note that both the key and the value type of the dictionary must be string. At first sight, such an extension method might seem of little help, because it just adds static data that can only be set at compile time. However, an in-memory collection still allows you to isolate parametric data and the provider model of the ASP.NET Core configuration API decouples that data from the main body of the application and injects it for you at startup, like so:

```
new ConfigurationBuilder()
  .AddInMemoryCollection(
    new Dictionary<string, string> {{"Timezone", "+1"}})
  .Build();
```

In the previous code snippet, for example, a value representing the timezone to use is appended to the configuration builder, and the rest of the application receives it through the unified interface of the configuration API. In this way, you don't have to change anything other than the provider and the actual storage pattern to read the timezone (as well as any other data you inject from memory) from other data sources.

Finally, the method AddEnvironmentVariables adds any environment variables defined in the server instance to the configuration tree. Note that all defined environment variables are added as a single block to the tree. If you need filtering, you're best off opting for an in-memory provider and copying only selected variables to the dictionary.

Note that in ASP.NET Core 2.0 you can also inject IConfiguration right in the constructor of the startup class, and have the configuration tree automatically configured with environment variables and content from two JSON files: appsettings.json and appsettings.development.json. If you want JSON files with a different name or another list of providers, just build the configuration tree from scratch as follows:

```
var dom = new ConfigurationBuilder()
  .SetBasePath(env.ContentRootPath)
  .AddJsonFile("MyAppSettings.json")
  .AddInMemoryCollection(new Dictionary<string, string> {{"Timezone", "+1"}})
  .Build();
```

Custom Configuration Providers

In addition to using predefined providers, you can also create your own configuration provider. To build a custom configuration provider, you start with a wrapper configuration source class—a plain class that implements IConfigurationSource. Here's an example:

```
public class MyDatabaseConfigSource : IConfigurationSource
{
  public IConfigurationProvider Build(IConfigurationBuilder builder)
  {
    return new MyDatabaseConfigProvider();
  }
}
```

In the implementation of the method Build, as shown in the previous code snippet, you finally reference the actual provider—namely a class that inherits from the system-defined ConfigurationProvider class (see **Figure 1**).

A common example is a configuration provider that reads from an ad hoc database table. The provider may ultimately hide the schema of the table and the layout of the database involved. The connection string, for example, might be a private constant value. Such a configuration provider would likely use Entity Framework (EF) Core to perform data access tasks and, therefore, needs to have available a dedicated DbContext class and a dedicated set of entity classes to fetch values to be later converted into a string/string dictionary. As a nice touch, you might want to define default values for any of the values expected to be found and populate the database, if empty.

The database-driven provider discussed here is closed around a well-known database table. However, if you find a way to pass a DbContextOptions object as an argument to the provider, you can even manage to work with a rather generic EF-based provider. An example of this technique can be found at bit.ly/2uQBJmK.

Figure 1 **Sample Database-Driven Configuration Provider**

```
public class MyDatabaseConfigProvider : ConfigurationProvider
{
  private const string ConnectionString = "...";

  public override void Load()
  {
    using (var db = new MyDatabaseContext(ConnectionString))
    {
      db.Database.EnsureCreated();
      Data = !db.Values.Any()
        ? GetDefaultValues(db)
        : db.Values.ToDictionary(c => c.Id, c => c.Value);
    }
  }

  private IDictionary<string, string> GetDefaultValues (MyDatabaseContext db)
  {
    // Pseudo code for determining default values to use
    var values = DetermineDefaultValues();

    // Save default values to the store
    // NOTE: Values is a DbSet<T> in the DbContext being used
    db.Values.AddRange(values);
    db.SaveChanges();

    // Return configuration values
    return values;
  }
}
```

Figure 2 **A Sample JSON File**

```
{
  "ApplicationTitle" : "Programming ASP.NET Core",
  "GeneralSettings" : {
    "CopyrightYears" : [2017, 2018],
    "Paging" : {
      "PageSize" : 20,
      "FreezeHeaders" : true
    },
    "Sorting" : {
      "Enabled" : true
    }
  }
}
```

Building the Configuration Tree

The resulting configuration tree—personally, I like to call it the configuration document object model—is commonly built in the constructor of the startup class. The output generated by the Build method of the ConfigurationBuilder class is an object of type IConfigurationRoot. The startup class will provide a member to save the instance for further use at a later time throughout the entire application stack, like this:

```
public class Startup
{
  public IConfigurationRoot Configuration { get; }
  public Startup(IHostingEnvironment env)
  {
    var dom = new ConfigurationBuilder()
      .SetBasePath(env.ContentRootPath)
      .AddJsonFile("MyAppSettings.json")
      .Build();
     Configuration = dom;
  }
}
```

The IConfigurationRoot object is the connection point for the application components to read the individual values in the configuration tree.

Reading Configuration Data

To read configuration data programmatically, you use the GetSection method on the IConfigurationRoot object. The value is read as a plain string. To identify the exact value you want to read, you provide a path string in which the colon symbol (:) is used to delimit properties in a hierarchical schema. Suppose that your ASP.NET Core project includes a JSON file like the one in **Figure 2**.

To read settings, you can proceed in many different ways, but it's mandatory that you know how to navigate to the actual value in the configuration tree. For example, to locate the place where the default size of the page is stored the path is:

```
Generalsettings:Paging:PageSize
```

Note that a path string is always case-insensitive. In light of this, the simplest way to read settings is through the indexer API, like this:

```
var pageSize = Configuration["generalsettings:paging:pagesize"];
```

It's important to note that the indexer API returns the value of the setting as a string, but you can also use an alternate strongly typed API. Here's how that approach looks:

```
var pathString = "generalsettings:paging:pagesize";
var pageSize = Configuration.GetValue<int>(pathString);
```

The reading API is independent of the actual data source. You use the same API to read hierarchical content from JSON as you do to read flat content from in-memory dictionaries.

In addition to direct reading, you can leverage a positioning API, which will conceptually move the reading cursor on a specific configuration subtree. The GetSection method lets you select an entire configuration subtree where you can act on using both the indexer and the strongly typed API. The GetSection method is a generic query tool for the configuration tree and isn't specific of JSON files only. An example is shown here:

```
var pageSize = Configuration.GetSection("Paging").GetValue<int>("PageSize");
```

Note that for reading you also have available a GetValue method and the Value property. Both would return the value of the setting as a plain string.

Refreshing Loaded Configuration

In ASP.NET Core, the configuration API is designed to be read-only. This only means that you can't write back to the configured data source using an official API. If you have a way to edit the content of the data source (that is, programmatic overwrites of text files, database updates and the like), then the system allows you reload the configuration tree programmatically.

To reload a configuration tree, all you need to do is call the Reload method in the configuration root object.

```
Configuration.Reload();
```

Typically, you might want to use this code from within an admin page where you offer users a form to update stored settings. As far as JSON files are concerned, you can also enable automatic reloads of the content upon changes. You just add an extra parameter to AddJsonFile, like so:

```
var dom = new ConfigurationBuilder()
  .AddJsonFile("MyAppSettings.json", optional: true, reloadOnChange: true);
```

JSON files are the most popular of the text file formats natively supported by ASP.NET Core. You can also load settings from XML and .ini files. You just add a call to AddXmlFile and AddIniFile methods with the same rich signature of AddJsonFile.

Note that in ASP.NET Core 2, configuration can also be managed right from the program.cs file, as shown here:

```
return new WebHostBuilder()
  .UseKestrel()
  .UseContentRoot(Directory.GetCurrentDirectory())
  .ConfigureAppConfiguration((builderContext, config) =>
  {
    var env = builderContext.HostingEnvironment;
    config.AddJsonFile("appsettings.json")
  });
```

If you do, then you can inject the configuration tree in the startup class via IConfiguration in the constructor.

Passing Configuration Data Around

The configuration root object lives within the scope of the startup class, but its content should be made available throughout the entire application. The natural way of achieving this goal in ASP.NET Core is via dependency injection (DI). To share the configuration root object with the system, you just bind it to the DI system as a singleton.

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddSingleton(Configuration);
  ...
}
```

After that, you can inject an IConfigurationRoot reference in any controller constructors and Razor views. Here's an example for a view:

```
@inject IConfigurationRoot Configuration
CURRENT PAGE SIZE IS @Configuration["GeneralSettings:Paging:PageSize"]
```

While injecting the configuration root object is possible, and to a large extent even easy, it still results in a fairly cumbersome API if access to configuration settings occurs too often. That's why the ASP.NET Core configuration API offers a serialization mechanism that lets you save the configuration tree, all or in part, to a separate POCO class.

Serializing to POCO Classes

If you've ever done any ASP.NET MVC programming, you should be familiar with the concept of model binding. A similar pattern—called the Options pattern—can be used in ASP.NET Core to do this, as shown here:

```
public void ConfigureServices(IServiceCollection services)
{
  // Initializes the Options subsystem
  services.AddOptions();
  // Maps the PAGING section to a distinct dedicated POCO class
  services.Configure<PagingOptions>(
    Configuration.GetSection("generalsettings:paging"));
}
```

In this example, you first initialize the configuration binding layer, as shown in the previous code snippet, and then explicitly ask to try to bind the content of the specified subtree to the public properties of the given class, like this:

```
public class PagingOptions
{
  public int PageSize { get; set; }
  public bool FreezeHeaders { get; set; }
}
```

The bound class must be a POCO class with public getter/setter properties and a default constructor. The Configure method will attempt to fetch and copy values from the specified section of the configuration tree right into a freshly created instance of the class. Needless to say, binding will silently fail if values aren't convertible to declared types.

Such a POCO class can be passed around throughout the application stack using the built-in DI system. You don't have to do anything for this to happen. Or rather, any configuration required is applied when you invoke AddOptions. Just one step remains to programmatically access the configuration data serialized to a class:

```
public PagingOptions PagingOptions { get; }
public CustomerController(IOptions<PagingOptions> config)
{
  PagingOptions = config.Value;
}
```

If you use the Options pattern extensively in all of your controllers, you might consider moving the options property (that is, PagingOptions) to a base class and then inherit your controller classes from there. Likewise, you can inject IOptions<T> in any Razor views.

Wrapping Up

In classic ASP.NET MVC, the best practice for dealing with configuration data mandates that you load all of your data once at startup into a global container object. The global object is then accessible from controller methods and its content can be injected as an argument into back-end classes such as repositories and even into views. In classic ASP.NET MVC, the cost of mapping loose string-based data into the strongly typed properties of the global container is entirely on you.

In ASP.NET Core, you have both a low-level API to read individual values in a much more precise way than the ConfigurationManager API of old ASP.NET, and an automatic way to serialize external content into POCO classes of your design. This is possible because the introduction of the configuration tree—populated by a list of providers—decouples configuration from the main body of the application. ■

Dino Esposito *is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Programming ASP.NET Core" (Microsoft Press, 2018). Pluralsight author and developer advocate at JetBrains, Esposito shares his vision of software on Twitter: @despos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Ugo Lattanzi*

TEST RUN

JAMES MCCAFFREY

Understanding k-NN Classification Using C#

The goal of a classification problem is to use the values of two or more predictor variables (often called features in machine learning [ML] terminology) to predict a class (sometimes called a label). For example, you might want to predict the risk of a server failing (low, medium, high) based on the average number of HTP requests handled and the physical temperature of the server.

There are many ML classification techniques, including naive Bayes classification, neural network classification, and decision tree classification. The k-nearest neighbors (k-NN) algorithm is a relatively simple and elegant approach. Relative to other techniques, the advantages of k-NN classification are simplicity and flexibility. The two primary disadvantages are that k-NN doesn't work well with non-numeric predictor values, and it doesn't scale well to huge data sets.

> Many ML techniques, including k-NN classification, require training data that has known, correct predictor values and class labels.

This article explains exactly how k-NN classification works and presents an end-to-end demo program written in C#. The best way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo problem is to predict the class ("0," "1," "2") of an item that has two predictor variables with values (5.25, 1.75). If k (the number of neighbor values to examine) is set to 1, then the predicted class is "1." If k is set to 4, the predicted class is "2." Behind the scenes, the demo program uses a set of 33 training data items to make the predictions.

Many ML libraries have built-in k-NN classification functions, but library functions can be difficult or impossible (due to legal issues) to integrate into a production system. Understanding how to implement k-NN classification from scratch gives you full control, and the ability to customize your system.

This article assumes you have intermediate or better programming skills, but doesn't assume you know anything about k-NN classification. The demo program is coded using C#, but you shouldn't have too much trouble refactoring the code to another language, such as Java or Python. The demo program is a bit too long to present in its entirety, but the complete source code is available in the file download that accompanies this article.

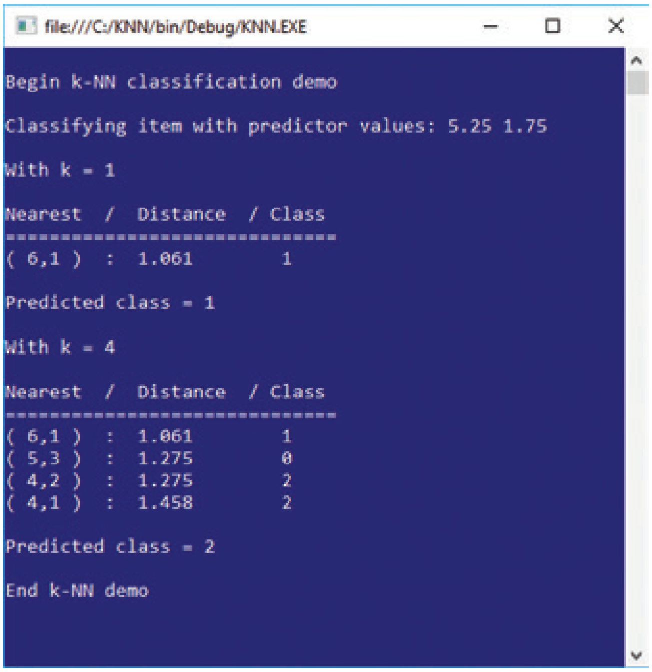
Code download available at msdn.com/magazine/1217magcode.

Understanding the k-NN Algorithm

The graph in **Figure 2** represents the data used in the demo program. There are 33 training items. Each item has two predictor values, x0 and x1. The k-NN algorithm can handle problems with any number of predictors, but the demo uses just two so that the problem can be easily visualized in a graph.

The values of the predictor variables are all between 0 and 9. When performing k-NN classification, it's important to normalize the predictor values so that large magnitudes don't overwhelm small magnitudes. For example, suppose you have predictor variables annual income (such as $48,000) and age (such as 32). You could normalize by dividing all income values by 10,000 (giving values like 4.8) and dividing all age values by 10 (giving values like 3.2). Two other common normalization techniques are z-score normalization and min-max normalization.

Many ML techniques, including k-NN classification, require training data that has known, correct predictor values and class



```
Begin k-NN classification demo

Classifying item with predictor values: 5.25 1.75

With k = 1

Nearest  /  Distance  / Class
==============================
( 6,1 )  :  1.061       1

Predicted class = 1

With k = 4

Nearest  /  Distance  / Class
==============================
( 6,1 )  :  1.061       1
( 5,3 )  :  1.275       0
( 4,2 )  :  1.275       2
( 4,1 )  :  1.458       2

Predicted class = 2

End k-NN demo
```

Figure 1 **Classification Demo Using the k-NN Algorithm**

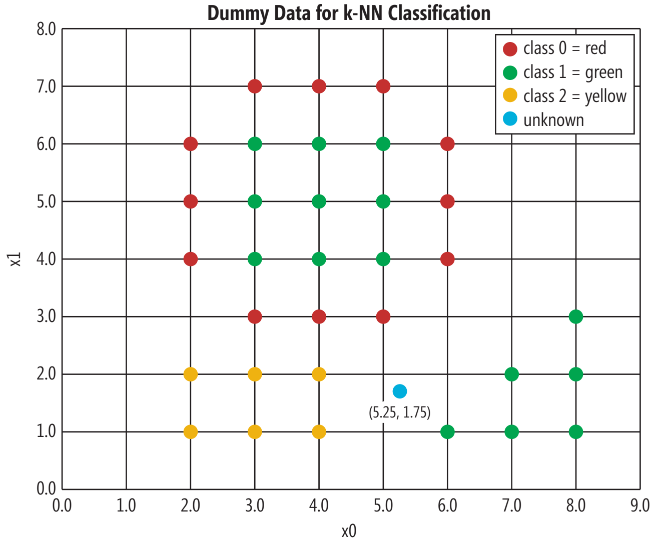


Figure 2 **Demo Program Training Data**

labels. The demo training data has three different classes, indicated by red, green and yellow. The blue data point at (5.25, 1.75) is the unknown to predict. When k is set to 1, k-NN finds the single, closest training data item and returns the class of that closest data item as the predicted class of the unknown item.

In this example, for the (5.25, 1.75) unknown item, the closest training data item is the green (class = “1”) dot at (6.0, 1.0) so the predicted class is “1.” When using k-NN, you must specify how to measure distance between data items so you can define what “closest” means. The demo program uses Euclidean distance. The distance between (5.25, 1.75) and (6.0, 1.0) is sqrt((5.25 - 6.0)^2 + (1.75 - 1.0)^2) = sqrt(0.5625 + 0.5625) = sqrt(1.1250) = 1.061, as displayed in **Figure 1**.

When k is set to 4, the predicted class depends on the four nearest training data items. In the demo example, the four closest items are at (6.0, 1.0), (5.0, 3.0), (4.0, 2.0) and (4.0, 1.0). The associated class labels for the items are (“1,” “0,” “2,” “2”). Because there are more “2” labels than “0” and “1” labels, the predicted class is “2.”

When using k-NN you must specify how to determine the predicted class from the set of the k closest class labels. The demo program uses a majority-vote approach. Notice that when using a simple majority vote, you can run into tie situations. In practice, however, k-NN majority-vote ties are relatively rare. The demo program returns the lowest class label in case of ties. For example, if the associated labels are (“2,” “1,” “1,” “2”), then the demo program voting mechanism would return “1.”

The Demo Program

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

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

The overall program structure, with a few minor edits to save space, is presented in **Figure 3**. The demo uses a simple static method approach, and all normal error checking has been removed to keep the main ideas clear.

The demo begins by setting up the training data:

```
static void Main(string[] args)
{
  Console.WriteLine(“Begin k-NN classification demo “);
  double[][] trainData = LoadData();
...
```

Figure 3 **Overall Program Structure**

```
using System;
namespace KNN
{
  class KNNProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin k-NN classification demo ");

      double[][] trainData = LoadData();

      int numFeatures = 2;
      int numClasses = 3;

      double[] unknown = new double[] { 5.25, 1.75 };
      Console.WriteLine("Predictor values: 5.25 1.75 ");

      int k = 1;
      Console.WriteLine("With k = 1");
      int predicted = Classify(unknown, trainData,
        numClasses, k);
      Console.WriteLine("Predicted class = " + predicted);

      k = 4;
      Console.WriteLine("With k = 4");
      predicted = Classify(unknown, trainData,
        numClasses, k);
      Console.WriteLine("Predicted class = " + predicted);

      Console.WriteLine("End k-NN demo ");
      Console.ReadLine();
    }

    static int Classify(double[] unknown,
      double[][] trainData, int numClasses, int k) { . . }

    static int Vote(IndexAndDistance[] info,
      double[][] trainData, int numClasses, int k) { . . }

    static double Distance(double[] unknown,
      double[] data) { . . }

    static double[][] LoadData() { . . }

  } // Program class

  public class IndexAndDistance : IComparable<IndexAndDistance>
  {
    public int idx;  // Index of a training item
    public double dist;  // To unknown

    // Need to sort these to find k closest
    public int CompareTo(IndexAndDistance other)
    {
      if (this.dist < other.dist) return -1;
      else if (this.dist > other.dist) return +1;
      else return 0;
    }
  }

} // ns
```

The data is hardcoded and stored into an array-of-arrays-style matrix. In a non-demo scenario you'd likely read data into memory from a text file. Because k-NN classification typically stores all training data into memory, the technique doesn't easily scale to scenarios with very large data sets. Method LoadData is defined as:

```
static double[][] LoadData()
{
  double[][] data = new double[33][];
  data[0] = new double[] { 2.0, 4.0, 0 };
    ...
  data[12] = new double[] { 3.0, 4.0, 1 };
    ...
  data[32] = new double[] { 4.0, 2.0, 2 };
  return data;
}
```

The first two values of each item are the predictor values and the last value is the class label. A common alternative design is to store predictor values in a matrix, but store the class labels in a separate array. The demo code assumes the class labels are numeric, and consecutively numbered starting at 0. If your class labels are non-numeric, such as "low," "medium," "high," you must encode the data, either in a preprocessing stage, or programmatically when reading the data into memory.

Next, the demo sets up the unknown item to predict, like so:

```
int numFeatures = 2;
int numClasses = 3;
double[] unknown = new double[] { 5.25, 1.75 };
Console.WriteLine("Predictor values: 5.25 1.75 ");
```

The demo program doesn't actually use the numFeatures variable, but as you'll see shortly, there are many possible customization points for k-NN classification, and a numFeatures value can be useful.

Next, the demo makes the simplest possible k-NN prediction:

```
int k = 1;
Console.WriteLine("With k = 1");
int predicted = Classify(unknown, trainData,
  numClasses, k);
Console.WriteLine("Predicted class = " + predicted);
```

The Classify method accepts parameters for the item to predict, a matrix of training data, the number of classes in the training data, and the number of nearest neighbors to evaluate. In principle, you could implement method Classify so that it scans the training data to programmatically determine the number of classes, but the effort required outweighs the benefit, in my opinion.

The demo program concludes with:

```
...
  k = 4;
  Console.WriteLine(“With k = 4”);
  predicted = Classify(unknown, trainData,
    numClasses, k);
  Console.WriteLine(“Predicted class = “ + predicted);

  Console.WriteLine(“End k-NN demo “);
  Console.ReadLine();
}
```

There aren't any good rules of thumb for determining the value of k when using k-NN classification. The value of k in k-NN classification is a hyperparameter and must be determined by intuition and experimentation.

Implementing the k-NN Algorithm

In very high-level pseudo-code, the k-NN classification algorithm used by the demo is:

```
Compute distances from unknown to all training items
Sort the distances from nearest to farthest
Scan the k-nearest items; use a vote to determine the result
```

The definition of method Classify begins by computing and storing into an array the distances between the unknown item to classify and all training items, as shown in **Figure 4**.

It's not enough to store and sort just the distances from the unknown item to each training item, because you need to know the class associated with each distance. The demo program defines a simple container class, named IndexAndDistance, that holds an index to a training item and the associated distance to the unknown item:

```
public class IndexAndDistance : IComparable<IndexAndDistance>
{
  public int idx;  // index of a training item
  public double dist;  // distance to unknown
  public int CompareTo(IndexAndDistance other) {
    if (this.dist < other.dist) return -1;
    else if (this.dist > other.dist) return +1;
    else return 0;
  }
}
```

> It's not enough to store and sort just the distances from the unknown item to each training item, because you need to know the class associated with each distance.

The class label is stored indirectly because if you know the index of a training item, you can look up the class label in the matrix of training data as the last cell in the row pointed to by the index. An alternative design is to store just the class label explicitly, but storing the training item index gives you more flexibility. Another alternative is to explicitly store both the index and the class label.

Because k-NN needs to sort the index-distance items to determine the k-nearest items, the IndexAndDistance definition implements the IComparable interface by defining a CompareTo method. Doing this allows you to automatically sort an array of IndexAndDistance objects. If you refactor the demo code to a programming language that doesn't support auto-sorting, you'll have to implement a custom sort method that operates on a structure, or implement a custom sort method that works with parallel arrays.

There are a few alternatives to sorting index-distance items—for example, using a heap data structure—but in my opinion the increased complexity outweighs any performance improvement you'd get.

Figure 4 **Definition of the Classify Method**

```
static int Classify(double[] unknown,
  double[][] trainData, int numClasses, int k)
{
  int n = trainData.Length;
  IndexAndDistance[] info = new IndexAndDistance[n];
  for (int i = 0; i < n; ++i) {
    IndexAndDistance curr = new IndexAndDistance();
    double dist = Distance(unknown, trainData[i]);
    curr.idx = i;
    curr.dist = dist;
    info[i] = curr;
  }
...
```

Figure 5 **The Vote Method**

```
static int Vote(IndexAndDistance[] info, double[][] trainData,
  int numClasses, int k)
{
  int[] votes = new int[numClasses];  // One cell per class
  for (int i = 0; i < k; ++i) {       // Just first k
    int idx = info[i].idx;            // Which train item
    int c = (int)trainData[idx][2];   // Class in last cell
    ++votes[c];
  }

  int mostVotes = 0;
  int classWithMostVotes = 0;
  for (int j = 0; j < numClasses; ++j) {
    if (votes[j] > mostVotes) {
      mostVotes = votes[j];
      classWithMostVotes = j;
    }
  }

  return classWithMostVotes;
}
```

After all training index-distance items are stored, they're sorted, and information for the k-nearest items is displayed:

```
Array.Sort(info);  // sort by distance
Console.WriteLine("Nearest / Distance / Class");
Console.WriteLine("==========================");
for (int i = 0; i < k; ++i) {
  int c = (int)trainData[info[i].idx][2];
  string dist = info[i].dist.ToString("F3");
  Console.WriteLine("( " + trainData[info[i].idx][0] +
    "," + trainData[info[i].idx][1] + " )  :  " +
    dist + "        " + c);
}
```

The class label, c, is extracted from cell [2] of a row of training data. This assumes there are two predictor variables. A superior approach would be to pass a numFeatures argument to method Classify, then access cell [numFeatures].

Displaying information about the k-nearest training items is optional, but it illustrates an advantage of k-NN classification compared to many other techniques. The k-NN algorithm is somewhat interpretable in the sense that you can determine exactly how an unknown item was classified. Some techniques, notably neural network classification, are difficult or impossible to interpret.

The Classify method concludes by scanning the k-nearest training items to determine a predicted class for the unknown item:

```
  ...
  int result = Vote(info, trainData, numClasses, k);
  return result;
} // Classify
```

Helper method Vote accepts the array of all index-distance items. An alternative approach is to pass just the first k cells of the array.

Distance and Voting

The Classify method calls helper methods Distance and Vote. Method Distance is defined as:

```
static double Distance(double[] unknown, double[] data)
{
  double sum = 0.0;
  for (int i = 0; i < unknown.Length; ++i)
    sum += (unknown[i] - data[i]) * (unknown[i] - data[i]);
  return Math.Sqrt(sum);
}
```

This is a simple implementation of Euclidean distance. Common alternatives you can consider include Manhattan distance, Mahalanobis distance and measures based on similarity, such as the radial basis function kernel. Because k-NN needs a notion of "nearest" and most distance metrics work with strictly numeric or strictly non-numeric data, k-NN classification is not well-suited for problems with mixed numeric and categorical data.

Helper method Vote is presented in **Figure 5**. Determining a consensus class label from k items is a bit trickier than you might guess. The demo uses the simplest approach, where each of the k-nearest training items gets one vote for its class label. This approach doesn't take into account the distance. A common alternative is to weight votes by distance so that training items that are closer to the unknown item have more influence.

The demo implementation doesn't explicitly deal with duplicate training-data items. Much real-life data doesn't have an excessive amount of duplicate data, but if your training data does include lots of duplicates, you might want to consider removing them.

Wrapping Up

The k-NN classification algorithm is arguably one of the simplest of all machine learning techniques. In addition to simplicity, k-NN classification can easily deal with multi-class problems (as opposed to some techniques that work easily only for binary classification). Another advantage is that k-NN can easily deal with data that has unusual characteristics, such as the dummy demo data that has several pockets of class labels. Some techniques, such as logistic regression and non-kernel support vector machines, can deal only with data that is linearly separable. Two other relative advantages of k-NN classification are flexibility of implementation, and interpretability of results.

> The k-NN algorithm is somewhat interpretable in the sense that you can determine exactly how an unknown item was classified.

On the negative side, k-NN classification doesn't work well with categorical data or mixed numeric and categorical data. The technique doesn't scale well to huge data sets. And k-NN classification is highly sensitive to the local geometry of the training data.

When I have a classification problem with strictly numeric predictor values, and a non-huge set of training data (for example, less than one million items), using k-NN is often my first approach. Then I'll try more sophisticated techniques, typically including neural network classification. In some situations, an ensemble approach that combines the results of k-NN classification with neural network classification can lead to a very robust and accurate prediction system. ■

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: *Chris Lee, Ricky Loynd, Ken Tran*

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

March 11 – 16, 2018
Bally's Hotel & Casino
Las Vegas

Respect the Past. Code the Future.

Visual Studio Live! (VSLive!™) Las Vegas, returns to the strip, March 11 – 16, 2018. During this intense week of developer training, you can sharpen your skills in everything from ASP.NET to Xamarin.

Plus, celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Experience the education, knowledge-share and networking at #VSLive25.

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION

VSLive! 1998

VSLive! 2017

SUPPORTED BY

Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA INC YOUR GROWTH. OUR BUSINESS.

DEVELOPMENT TOPICS INCLUDE:

- VS2017/.NET
- Angular/JavaScript
- ASP.NET Core
- Xamarin
- Azure / Cloud
- Hands-On Labs
- Software Practices
- ALM / DevOps
- SQL Server 2017
- UWP (Windows)

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.

Register to code with us today!

Register by December 15 and Save $500!

Use promo code VSLDEC4

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/lasvegasmsdn

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

AGENDA AT-A-GLANCE

BACK BY POPULAR DEMAND

Sunday Pre-Con Hands-On Labs

Choose From:

HOL01 Special 2-Day Hands-On Lab: Modern Security Architecture for ASP.NET Core

Sunday, March 11,
9:00am – 6:00pm (Part 1)*

Monday, March 12,
9:00am – 6:00pm (Part 2)*

Brock Allen

You will learn:

- The security architecture of ASP.NET Core
- About authenticating users with OpenID Connect
- How to protect Web APIs with OAuth2

*This 2-day Hands-On Lab is available with the six-day conference package or on its own. Details at **vslive.com/lasvegasmsdn**.

HOL02 From 0-60 in a Day with Xamarin and Xamarin.Forms

Introductory / Intermediate

Sunday, March 11,
9:00am – 6:00pm

Roy Cornelissen & Marcel de Vries

You will learn:

- How to build your first mobile apps on three platforms with the Xamarin framework
- How to maintain platform uniqueness while sharing a large chunk of your codebase
- How to think “mobile first” in your application architecture

HOL03 Busy Developer’s HOL on Angular

Sunday, March 11,
9:00am – 6:00pm

Ted Neward

In this Hands-On Lab, we’ll start from zero, with a little TypeScript, then start working with Angular 2: its core constructs and how it works with components, modules, and of course the ubiquitous model/view/controller approach.

ONLY $595 through December 15
Applies to HOL02 and HOL03 only.

ALM / DevOps	Cloud Computing	Database and Analytics	Native Client

START TIME	END TIME	**Full Day Hands-On Labs: Sunday, March 11, 2018** *(Separate entry fee required)*	
8:00 AM	9:00 AM	Pre-Conference Hands-On Lab Registration - Coffee and Morning Pastries	
9:00 AM	6:00 PM	**HOL01** Full Day Hands-On Lab: Modern Security Architecture for ASP.NET Core (Part 1) *- Brock Allen*	
4:00 PM	6:00 PM	Conference Registration Open	
START TIME	END TIME	**Pre-Conference Workshops: Monday, March 12, 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	**HOL01** Full Day Hands-On Lab: Modern Security Architecture for ASP.NET Core (Part 2) *- Brock Allen*	
7:00 PM	9:00 PM	Dine-A-Round	
START TIME	END TIME	**Day 1: Tuesday, March 13, 2018**	
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries	
8:00 AM	9:15 AM	**T01** Go Serverless with Azure Functions *- Eric D. Boyd*	**T02** Getting Ready to Write Mobile Applications with Xamarin *- Kevin Ford*
9:30 AM	10:45 AM	**T05** Angular 101 *- Deborah Kurata*	**T06** Lessons Learned from Real World Xamarin.Forms Projects *- Nick Landry*
11:00 AM	12:00 PM	**KEYNOTE:** .NET Everywhere and for Everyone *- James Montemagno, Principal*	
12:00 PM	1:00 PM	Lunch	
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors	
1:30 PM	2:45 PM	**T09** Busy Developer’s Guide to Chrome Development *- Ted Neward*	**T10** Works On My Machine... Docker for Developers *- Chris Klug*
3:00 PM	4:15 PM	**T13** Angular Component Communication *- Deborah Kurata*	**T14** Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience *- Laurent Bugnion*
4:15 PM	5:30 PM	Welcome Reception	
START TIME	END TIME	**Day 2: Wednesday, March 14, 2018**	
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries	
8:00 AM	9:15 AM	**W01** The Whirlwind Tour of Authentication and Authorization with ASP.NET Core *- Chris Klug*	**W02** Building Mixed Reality Experiences for HoloLens & Immersive Headsets in Unity *- Nick Landry*
9:30 AM	10:45 AM	**W05** TypeScript: The Future of Front End Web Development *- Ben Hoelting*	**W06** A Dozen Ways to Mess Up Your Transition From Windows Forms to XAML *- Billy Hollis*
11:00 AM	12:00 PM	**General Session:** To Be Announced *- Kasey Uhlenhuth, Program Manager, .NET &*	
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: 0-60 for Small Projects in Visual Studio Team Services *- Alex Mullans*	
2:00 PM	2:20 PM	**W12** Fast Focus: HTTP/2: What You Need to Know *- Robert Boedigheimer*	
2:30 PM	3:45 PM	**W15** Advanced Fiddler Techniques *- Robert Boedigheimer*	**W16** Building Cross-Platforms Business Apps with C# and CSLA .NET *- Rockford Lhotka*
4:00 PM	5:15 PM	**W19** Assembling the Web - A Tour of WebAssembly *- Jason Bock*	**W20** Radically Advanced XAML: Dashboards, Timelines, Animation, and More *- Billy Hollis*
START TIME	END TIME	**Day 3: Thursday, March 15, 2018**	
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries	
8:00 AM	9:15 AM	**TH01** ASP.NET Core 2 For Mere Mortals *- Philip Japikse*	**TH02** Performance in 60 Seconds – SQL Tricks Everybody MUST Know *- Pinal Dave*
9:30 AM	10:45 AM	**TH05** Getting to the Core of ASP.NET Core Security *- Adam Tuliper*	**TH06** Secrets of SQL Server - Database Worst Practices *- Pinal Dave*
11:00 AM	12:00 PM	Panel Discussion: To Be Announced	
12:00 PM	1:00 PM	Lunch	
1:00 PM	2:15 PM	**TH09** Entity Framework Core 2 For Mere Mortals *- Philip Japikse*	**TH10** SQL Server 2017 - Intelligence Built-in *- Scott Klein*
2:30 PM	3:45 PM	**TH13** MVVM and ASP.NET Core Razor Pages *- Ben Hoelting*	**TH14** Introduction to Azure Machine Learning *- James McCaffrey*
4:00 PM	5:15 PM	**TH17** Securing Web Apps and APIs with IdentityServer *- Brian Noyes*	**TH18** Introduction to the CNTK v2 Machine Learning Library *- James McCaffrey*
START TIME	END TIME	**Post-Conference Workshops: Friday, March 16, 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	**F01** Workshop: Creating Mixed Reality Experiences for HoloLens & Immersive Headsets with Unity *- Nick Landry & Adam Tuliper*	

Speakers and sessions subject to change

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION

March 11 – 16, 2018
Bally's Hotel & Casino

LasVegas

Software Practices	Visual Studio / .NET Framework	Web Client	Web Server

<table>
<tr><td colspan="2"><b>Full Day Hands-On Labs: Sunday, March 11, 2018</b> <i>(Separate entry fee required)</i></td></tr>
<tr><td colspan="2">Pre-Conference Hands-On Lab Registration - Coffee and Morning Pastries</td></tr>
<tr><td><b>HOL02</b> Full Day Hands-On Lab: From 0-60 in a Day with Xamarin and Xamarin.Forms <i>- Roy Cornelissen & Marcel de Vries</i></td><td><b>HOL03</b> Full Day Hands-On Lab: Busy Developer's HOL on Angular <i>- Ted Neward</i></td></tr>
<tr><td colspan="2">Conference Registration Open</td></tr>
<tr><td colspan="2"><b>Pre-Conference Workshops: Monday, March 12, 2018</b> <i>(Separate entry fee required)</i></td></tr>
<tr><td colspan="2">Pre-Conference Workshop Registration - Coffee and Morning Pastries</td></tr>
<tr><td><b>M02</b> Workshop: Developer Dive into SQL Server 2016 <i>- Leonard Lobel</i></td><td><b>M03</b> Workshop: Add Intelligence to Your Solutions with AI, Bots, and More <i>- Brian Randell</i></td></tr>
<tr><td colspan="2">Dine-A-Round</td></tr>
<tr><td colspan="2"><b>Day 1: Tuesday, March 13, 2018</b></td></tr>
<tr><td colspan="2">Registration - Coffee and Morning Pastries</td></tr>
<tr><td><b>T03</b> Database Development with SQL Server Data Tools <i>- Leonard Lobel</i></td><td><b>T04</b> What's New in Visual Studio 2017 for C# Developers <i>- Kasey Uhlenhuth</i></td></tr>
<tr><td><b>T07</b> Introduction to Azure Cosmos DB <i>- Leonard Lobel</i></td><td><b>T08</b> Using Visual Studio Mobile Center to Accelerate Mobile Development <i>- Kevin Ford</i></td></tr>
<tr><td colspan="2"><i>Program Manager – Xamarin, Microsoft</i></td></tr>
<tr><td colspan="2">Lunch</td></tr>
<tr><td colspan="2">Dessert Break - Visit Exhibitors</td></tr>
<tr><td><b>T11</b> DevOps for the SQL Server Database <i>- Brian Randell</i></td><td><b>T12</b> To Be Announced</td></tr>
<tr><td><b>T15</b> PowerShell for Developers <i>- Brian Randell</i></td><td><b>T16</b> To Be Announced</td></tr>
<tr><td colspan="2">Welcome Reception</td></tr>
<tr><td colspan="2"><b>Day 2: Wednesday, March 14, 2018</b></td></tr>
<tr><td colspan="2">Registration - Coffee and Morning Pastries</td></tr>
<tr><td><b>W03</b> Using Feature Toggles to Separate Releases from Deployments <i>- Marcel de Vries</i></td><td><b>W04</b> Lock the Doors, Secure the Valuables, and Set the Alarm <i>- Eric D. Boyd</i></td></tr>
<tr><td><b>W07</b> Overcoming the Challenges of Mobile Development in the Enterprise <i>- Roy Cornelissen</i></td><td><b>W08</b> Computer, Make It So! <i>- Veronika Kolesnikova & Willy Ci</i></td></tr>
<tr><td colspan="2"><i>Visual Studio, Microsoft</i></td></tr>
<tr><td colspan="2">Birds-of-a-Feather Lunch</td></tr>
<tr><td colspan="2">Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)</td></tr>
<tr><td><b>W10</b> Fast Focus: Cross Platform Device Testing with xUnit <i>- Oren Novotny</i></td><td><b>W11</b> Fast Focus: Understanding .NET Standard <i>- Jason Bock</i></td></tr>
<tr><td><b>W13</b> Fast Focus: Serverless Computing: Azure Functions and Xamarin in 20 minutes <i>- Laurent Bugnion</i></td><td><b>W14</b> Fast Focus: TBD <i>- Scott Klein</i></td></tr>
<tr><td><b>W17</b> Versioning NuGet and npm Packages <i>- Alex Mullans</i></td><td><b>W18</b> Getting to the Core of .NET Core <i>- Adam Tuliper</i></td></tr>
<tr><td><b>W21</b> Encrypting the Web <i>- Robert Boedigheimer</i></td><td><b>W22</b> Porting MVVM Light to .NET Standard: Lessons Learned <i>- Laurent Bugnion</i></td></tr>
<tr><td colspan="2"><b>Day 3: Thursday, March 15, 2018</b></td></tr>
<tr><td colspan="2">Registration - Coffee and Morning Pastries</td></tr>
<tr><td><b>TH03</b> Demystifying Microservice Architecture <i>- Miguel Castro</i></td><td><b>TH04</b> Cognitive Services in Xamarin Applications <i>- Veronika Kolesnikova</i></td></tr>
<tr><td><b>TH07</b> Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself <i>- Jeremy Clark</i></td><td><b>TH08</b> Publish Your Angular App to Azure App Services <i>- Brian Noyes</i></td></tr>
<tr><td colspan="2">Panel Discussion: To Be Announced</td></tr>
<tr><td colspan="2">Lunch</td></tr>
<tr><td><b>TH11</b> Writing Testable Code and Resolving Dependencies - DI Kills Two Birds with One Stone <i>- Miguel Castro</i></td><td><b>TH12</b> Signing Your Code the Easy Way <i>- Oren Novotny</i></td></tr>
<tr><td><b>TH15</b> "Doing DevOps" as a Politically Powerless Developer <i>- Damian Brady</i></td><td><b>TH16</b> Analyzing Code in .NET <i>- Jason Bock</i></td></tr>
<tr><td><b>TH19</b> I'll Get Back to You: Task, Await, and Asynchronous Methods <i>- Jeremy Clark</i></td><td><b>TH20</b> Multi-targeting the World: A Single Project to Rule Them All <i>- Oren Novotny</i></td></tr>
<tr><td colspan="2"><b>Post-Conference Workshops: Friday, March 16, 2018</b> <i>(Separate entry fee required)</i></td></tr>
<tr><td colspan="2">Post-Conference Workshop Registration - Coffee and Morning Pastries</td></tr>
<tr><td><b>F02</b> Workshop: Distributed Cross-Platform Application Architecture <i>- Jason Bock & Rockford Lhotka</i></td><td><b>F03</b> Workshop: UX Design for Developers: Basics of Principles and Process <i>- Billy Hollis</i></td></tr>
</table>

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

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

REGISTER NOW

vslive.com/lasvegasmsdn

DON'T GET ME STARTED

DAVID S. PLATT

Crushing It

"Don't leave us in suspense, Plattski," readers begged me. I'd written two pieces (msdn.com/magazine/mt808507 and msdn.com/magazine/mt809122) about how my Advanced UX class had designed a mobile app for Isaac "Zak" Kohane, M.D., Ph.D., who is chairman of Bioinformatics at Harvard Medical School. He'd had some success providing precision care to his mother, who suffers from congestive heart failure (CHF). He used a Fitbit Aria scale to monitor her weight via the Web, so he could detect excess water retention and intervene before it degraded her heart function. The system worked, but he wanted a mobile app that would be easier to use and would allow him to store notes about his clinical encounters. (See his article at wbur.fm/2yLf2SR.)

"It was a great UX project," my readers asked, "but did you just leave it as a bundle of paper?"

Heck, no. Zak's vision made for an ideal project for my Harvard Summer Session class on Xamarin Forms. I challenged my students to take the sketches my earlier class had developed and build them into the mobile app that Zak wanted.

Xamarin Forms is a front-end toolkit that allows developers to write a single C# code base and run it on iOS and Android and Universal Windows Platform (UWP). Drag a button onto a form, and Xamarin automatically renders it in iPhone format when it runs on an iPhone, and in Android format when it runs on an Android device. It's not perfect, and it's not without growing pains, but in my opinion it has passed the tipping point at which it beats the alternatives. And it will only get better from here (I hope).

I used Microsoft HealthVault for the back end. It's a free medical database that lives in the cloud. Any authorized app can use it (see bit.ly/2zKcjHL). I especially needed its ability to upload data from smart devices, such as Zak's mother's weight scale. It worked well for us.

My students came from all over—from Romania, Moldova and Turkey; from, Brazil and India; from Texas and California. I insisted that my classroom be neutral territory—anyone needing to fight had to take it outside. But they didn't. It was beautiful to see the technical challenge enthrall them, to see them rise to it, to watch the Microsoft guy and the Amazon guy work together to blow this project away. Every day I exhorted them, "Come on you guys, 3,000 people got admitted to the hospital for CHF just *today*! They're counting on us, let's get this done."

Most students came from industry, but one young lady was an undergraduate junior in Computer Science. You know Computer Science degrees—theories of compiler design, Turing-complete algorithms, abstract stuff like that. Nothing in it had prepared her for the howling chaos that is modern software development. Instead of turning up her nose, she dove right in and sweated blood alongside us Neanderthal geeks, earning her A grade. I told her: "Sena, you've been bitten by the werewolf. You're one of us now. Welcome, and may God have mercy on you." I wonder how she's buckling down to her theory classes this fall.

We dove into another killer three-week session, like last January—staying late every night, then taking work home; banging, banging, banging to get all these pieces to work together in ways that they hadn't previously and didn't quite want to. For example, HealthVault's client SDK used .NET Standard, while Xamarin was just then acquiring this capability. We never could have succeeded without the brainpower and commitment of these particular students.

"Better Doctors"

Zak came to the final presentations, as did Miguel de Icaza, Xamarin's co-founder and CTO. Zak loved seeing these great tools—as he put it, "Something like this would make us better doctors." Miguel loved seeing his creation helping people. And I loved showing off what my students had accomplished. Here's a video look for you: youtu.be/379YjTKda7o.

So where to now? Zak has been using the app and we'll soon see what changes he needs. We're hoping to interest a dozen or so other doctors for a quick trial in the spring. Let me know if you want to take part, or know anyone who does. We'll present the results in June at Zak's annual Precision Medicine conference (bit.ly/2y6s26S), and go from there.

Ultimately, I'd love to make this an open source project. But making things free costs money. How about Microsoft sponsors it equally from the HealthVault and Xamarin units? Zak will bring the medical firepower. And Microsoft can pay me to manage it. It'll be a flagship project for all concerned. How about it, Microsoft? ■

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.*

Xamarin
+
Syncfusion®