THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 15, 2018 VOL 33 NO 13

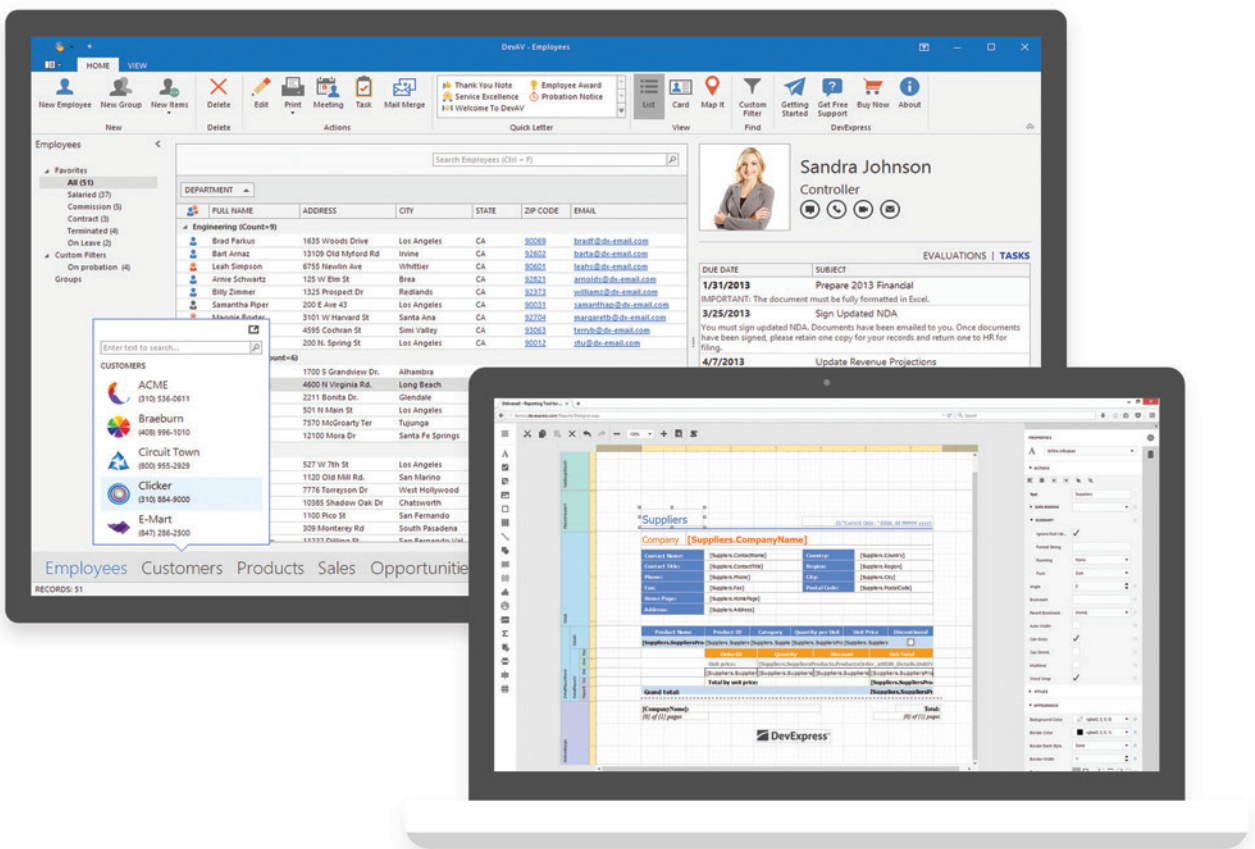
msdn magazine

Special Issue

DevExpress®

Your Next Great App Starts Here

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



Experience the DevExpress Difference
Download Your Free 30-Day Trial Today

devexpress.com/trial

THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 15, 2018 VOL 33 NO 13

msdn magazine

Special Issue

Connect(); Special Issue

Microsoft

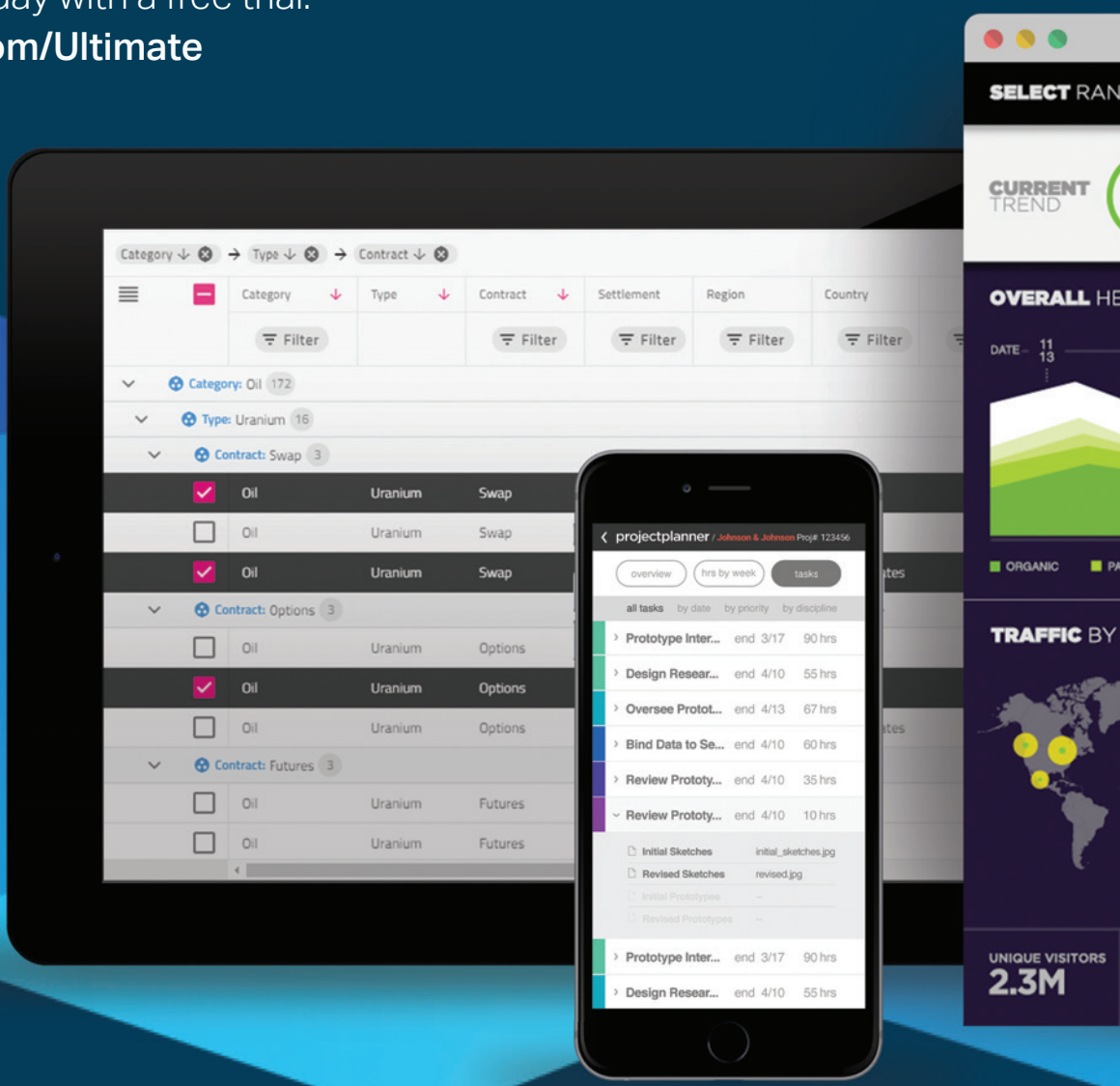
INFRAGISTICS

Faster Paths to Amazing Experiences

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

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

Get started today with a free trial:
Infragistics.com/Ultimate



To speak with sales or request a product demo with a solutions consultant call 1.800.231.8588

New Release

Infragistics Ultimate 18.2

- ✓ Fastest **grids** & **charts** on the market for the Angular developer
- ✓ The most complete Microsoft Excel and Spreadsheet solution for creating dashboards and reports without ever installing Excel
- ✓ An end-to-end design-to-code platform with Indigo.Design
- ✓ Best-of-breed charts for financial services

msdn magazine

DECEMBER 15, 2018 VOLUME 33 NUMBER 13

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 6300 Canoga Avenue, Suite 1150, Woodland Hills, CA 91367. Periodicals postage paid at Woodland Hills, CA 91367 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 866-293-3194 or 847-513-6011 for U.S. & Canada; 00-1-847-513-6011 for International, fax 847-763-9564. 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, Inc.
6300 Canoga Avenue, Suite 1150, Woodland Hills 91367
www.1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@Converge360.com

REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International
Phone: 212-221-9595
E-mail: 1105reprints@parsintl.com
www.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;
Email: jlong@meritdirect.com;
Web: www.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
2121 Alton Pkwy., Suite240, Irvine, CA 92606

Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
6300 Canoga Ave., Suite 1150, Woodland Hills, CA 91367

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

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®
Cloud Services

Introducing LEADTOOLS Cloud Services

LEADTOOLS Cloud Services is a high-powered and scalable Web API that gives developers a hassle-free interface for implementing OCR, Document Conversion, MICR, and Barcode into any application.

Get Started Today

GET 50 FREE PAGES WHEN YOU SIGN UP

SERVICES.LEADTOOLS.COM

MICHAEL DESMOND

December Is for Developers

Microsoft Connect(); has emerged as one of the most important and impactful developer-oriented events on the Microsoft calendar. Since 2014, Microsoft has used Connect(); to evangelize its growing cross-platform and open source software development efforts, release important products like Visual Studio for Mac, and articulate strategies in areas like artificial intelligence (AI) and the Internet of Things.

This year Connect(); kicked off on Dec. 4, and the conference hit on all these notes and more. On the open source front, Microsoft and Docker announced the Cloud Native Application Bundle (CNAB)—an open source, cloud-agnostic specification for packaging and running distributed applications. Microsoft also announced the open sourcing of Windows Presentation Foundation, Windows Forms and the Windows UI XAML Library.

> Connect(); is part of a larger conversation between Microsoft and its developers.

There was plenty of tool news on tap. Microsoft debuted Visual Studio 2019 Preview, and revealed new capabilities like Visual Studio Live Share for remote collaboration and Visual Studio IntelliCode, which expands language support to XAML and C++ and leverages AI for improved code insight. Also announced were the general availability of .NET Core 2.2, and the release of the public preview of .NET Core 3.0. Cross-platform development got a boost as well, with the Xamarin.Forms 4.0 Public Preview and the general availability of Xamarin.Forms 3.4.

Of course, AI and machine learning (ML) have been a huge area of focus at Microsoft. Connect(); this year provided a platform for some key announcements, including the general availability of Azure Machine Learning Service, which helps developers and data scientists quickly build, train and deploy ML models. Also released was the public preview of ML.NET, Microsoft's open source, cross-platform ML framework that helps developers infuse AI into their .NET applications.

This special issue of *MSDN Magazine* is dedicated to helping developers take full advantage of the tools and technologies featured at the Connect(); conference. Articles like Scott Hunter's "What's Coming in .NET Core 3.0," Julie Lerman's "Collaborative Development with Visual Studio Live Share" and Mads Kristensen's "What's New in Visual Studio 2019" will help you get a jump on the improvements to these flagship tools and frameworks.

DevOps has been a core message at Connect(); over the years and is the focus of the article "Deploy Your Code the Right Way with Azure Pipelines," written by Micheal Learned and Andy Lewis. And don't miss the pair of features exploring ML. James McCaffrey offers an introduction to the ML.NET framework, while Krishna Anumalasetty explores the automated ML capabilities of Azure Machine Learning and how they make AI accessible to more organizations.

There's more still, including David Ortinau's article, "Exploring the Xamarin.Forms Shell," that shows exciting new capabilities coming in Xamarin.Forms 4.0, and Michael Crumb's productivity-minded piece titled "7 Tips and Tricks for Azure App Service." Finally, be sure to check out our bonus Web feature from Kevin Farlee, "Introducing Azure SQL Database Hyperscale," which explores the re-architected storage engine in the SQL database and how it enables a highly scalable service tier for databases that adapt to workloads on-demand (msdn.com/magazine/mt848637).

Connect(); is part of a larger conversation between Microsoft and its developers. It's an opportunity for Microsoft to articulate its vision, and for developers to provide feedback that shapes the direction of that vision going forward. Learn more at microsoft.com/connectevent.

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

Switch to Amyuni® PDF

AMYUNI

Create and Process PDFs in .NET, COM/ActiveX and UWP

NEW v6.0

- Edit, process and print PDF 2.0 documents
- Create, fill-out and annotate PDF forms with Javascript support
- Fast and lightweight 64-bit components
- Universal Apps DLLs enable publishing C#, C++, CX or JS apps to Windows Store
- Plus a number of exciting features in v6.0

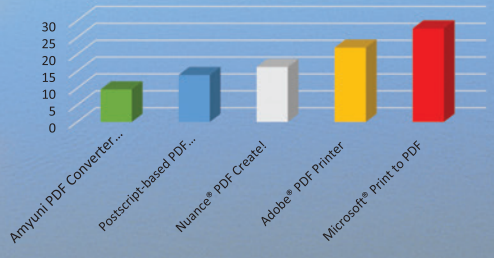
High Performance PDF Printer for Desktops and Servers

Print to PDF in a fraction of the time needed with other tools. WHQL certified for all Windows platforms. Version 6.0 updated for Server 2016.

Complete Suite of PDF, XPS and DOCX Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment
- Includes our Microsoft® WHQL certified PDF and DOCX printer driver
- Export PDF documents into other formats such as images, HTML5 or editable DOCX
- Import/Export XPS files using our native libraries

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



Other Developer Components from Amyuni®

- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract 3 engine for accurate character recognition
- Javascript Engine: Integrate a full JS interpreter into your PDF processing applications
- USB Mobile Monitor: Mirror the display of your Windows or Linux system onto your Android device

Windows Server 2016 Certified

Windows Server 2012 Certified

Windows

All development tools available at
www.amyuni.com

AMYUNI Technologies

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

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

What's Coming in .NET Core 3.0

Scott Hunter

.NET Core 3.0 is the next major version of the .NET Core platform. This article walks through the history of .NET Core and demonstrates how it has grown from basic support for Web and data workloads in version 1 to being able to run Web, desktop, machine learning, containers, IoT and more in version 3.0.

.NET Core 1

The .NET Core journey began a few years ago, with version 1 in 2016, with the goal of building the first version of .NET that was open source and cross-platform (Windows, macOS and Linux). This was driven by customers who could only use frameworks that were open source and by other customers who needed their .NET applications to run on Linux servers. Because .NET Core is cross-platform, it was designed so everything could be done from the command line, without the need for an IDE. And learning from the compatibility challenges of one globally installed .NET Framework, it was designed with side-by-side support, including shipping the framework as part of the application so the application doesn't depend on any framework being installed on the machine. Version 1 shipped with new versions of ASP.NET and Entity Framework (EF) and primarily targeted Web applications.

This article discusses technologies that are in preview. All information is subject to change.

This article discusses:

- .NET Core, versions 1, 2 and 3.0
- ASP.NET Core
- Entity Framework Core
- .NET Standard 2.1
- C# 8.0

Technologies discussed:

.NET Core, .NET Standard, C#

.NET Core 2

While version 1 got .NET running on new platforms, it supported only a limited set of .NET APIs. In order to address this, we created .NET Standard, which specified the APIs that any .NET runtime must implement so that code and binaries can be shared across .NET platforms and versions. With .NET Standard 2.0, we added more than 20,000 APIs to the .NET Standard spec. Version 2 of .NET Core shipped in June 2017 and included support for .NET Standard 2.0, giving it access to those APIs. We also introduced the Windows Compatibility Pack, which is a NuGet package that includes many Windows-only APIs, such as System.Drawing, System.DirectoryServices and more. ASP.NET Core 2.0 brought Razor Pages and SignalR, two frameworks that were missing from .NET Core 1.0. Entity Framework Core added support for lazy loading, a popular feature from Entity Framework. .NET Core 2 also continued the push to make .NET one of the fastest full-stack frameworks. The TechEmpower benchmark, which is run by an

independent company, lists .NET Core as No. 7 in raw plaintext performance and No. 6 in the Fortunes test of Web and data performance, beating Java servlet and Node.js (bit.ly/2PEE1l1).

.NET Core 3.0

.NET Core 3.0 is the next major version of the .NET Core platform. It includes many exciting new features, such as support for Windows desktop applications with Windows Forms (WinForms), Windows Presentation Foundation (WPF) and Entity Framework 6. For Web development it adds support for building client-side Web applications with C# using Razor Components (formerly known as Blazor). And it includes support for C# 8.0 and .NET Standard 2.1.

> For the first time ever, customers will be able to see the open development of these frameworks and can even help by filing issues, fixing bugs or helping develop new features live in GitHub.

We are adding support for Internet-of-Things (IoT) scenarios with .NET Core 3.0. You'll now be able to program hardware pins (for controlling devices and reading sensor data) on the Raspberry Pi and similar devices, and communicate via serial port on all supported OSes (for example, with a Raspberry Pi or Arduino). We're also adding IoT device support for ARM64 in this release, to complement the ARM32 capability that's already in place.

.NET Core 3.0 will also fully support ML.NET, our open source machine learning framework built for .NET developers. ML.NET powers products like Azure Machine Learning, Windows Defender and PowerPoint Design Ideas. Using ML.NET you can add many popular machine learning scenarios to your apps like sentiment analysis, recommendations, forecasting, image classification and more. Learn more at bit.ly/2OLRGRQ.

We recently released the first preview of .NET Core 3.0. For more information about .NET Core 3.0 and to try out the preview, see aka.ms/netcore3preview1.

Desktop (WinForms and WPF) and Open Source

WinForms and WPF are two of the most popular .NET application types and are used by millions of developers. .NET Core 3.0 adds support for WinForms and WPF, bringing Windows desktop development to .NET Core. .NET Core has always been about open source, and both frameworks will be open source in GitHub with the rest of .NET Core. For the first time ever, customers will be able to see the open development of these frameworks and can even help by filing issues, fixing bugs or helping develop new features live in GitHub. WinUI XAML Library will also be open sourced, and with XAML Islands you'll be able to use these controls in WinForms and WPF applications.

Many of the existing WinForms and WPF applications use Entity Framework to access data, so Entity Framework 6 will also be supported on .NET Core.

You might wonder why you'd want to build desktop applications on .NET Core. That's easy: It will give you access to all the advancements in .NET Core. You can build applications on the latest version of the framework without having to install .NET Core, and you can publish both your application and .NET Core into a single .EXE. .NET Core was designed with side-by-side in mind, so you can have multiple versions on a computer and applications can be locked to the version for which they were designed. And because of this side-by-side nature, the APIs in .NET Core, including WinForms and WPF, can be improved without the risk of breaking applications.

ASP.NET Core 3

.NET Core 3.0 is not all about the desktop, however. There are lots of exciting new features designed for the Web, as well. Let's take a look at a few of the features on which we're working.

A common question from customers is how to have an RPC (as in .NET Remoting and Windows Communication Foundation) experience on .NET Core. We are contributing to the gRPC (grpc.io) project to ensure gRPC will have first-class support for .NET developers.

Earlier this year we started an experiment in client-side Web development using .NET that we call Blazor. Blazor enables you to write Web UI components that run directly in the browser on a WebAssembly-based .NET runtime without writing a single line of JavaScript. You author components using Razor syntax, which are then compiled along with your code into normal .NET assemblies.

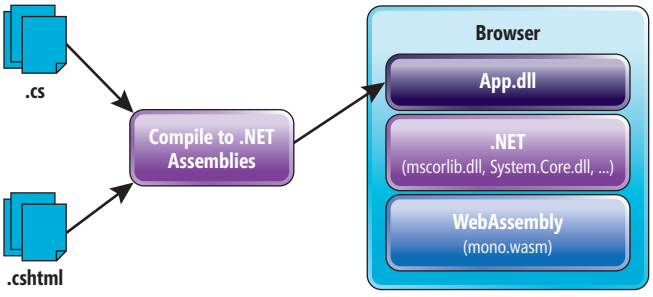


Figure 1 **Client-Side Web Development with Blazor**

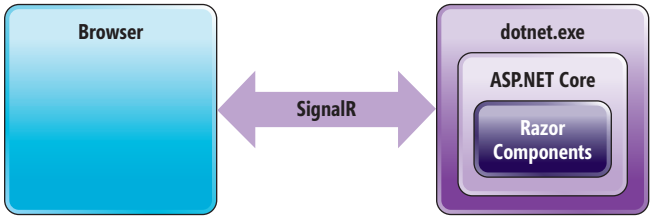


Figure 2 **Running UI Web Components on the Server Using SignalR**

The assemblies and the WebAssembly-based .NET runtime are then downloaded into the browser and executed using only open Web standards (no plug-ins or code transpilation required), as shown in **Figure 1**.

Alternatively, the same components can be run on the server using .NET Core, where all UI interactions and DOM updates are handled over a SignalR connection, as shown in **Figure 2**. When the components execute, they track what updates are required to the DOM and send these updates to the browser over the SignalR connection to be applied. UI events are sent to the server using the same connection. This model has several advantages: The download size is much smaller, your code is centralized on the server, and you get all the features and performance benefits of running on .NET Core.

We've been working on a Cosmos DB provider for EF Core, to enable developers familiar with the EF programing model to easily target Azure Cosmos DB as an application database.

For .NET Core 3.0 we're integrating the Blazor component model into ASP.NET Core. We call this integrated component model Razor Components. Razor Components enable a new era of composable UIs with ASP.NET Core, and full-stack Web development with .NET. Initially for .NET Core 3.0, Razor Components will run on the server, either as standalone routable components or used from Razor Pages and Views. The *same* components, however, can also be run client side on WebAssembly. In parallel with the .NET Core 3.0 work, we'll continue work on supporting Razor Components on WebAssembly using the interpreter-based .NET runtime, which we expect to ship in a subsequent release. Later, we also plan to release support for full ahead-of-time compilation of .NET code to WebAssembly, which will bring significant improvements to runtime performance.

EF Core 3.0

LINQ is a beloved .NET feature that enables you to write database queries without leaving your language of choice, taking advantage of rich type information to get IntelliSense and compile-time type checking. But LINQ also enables you to write a virtually unlimited number of complicated queries, and that has always been a huge challenge for LINQ providers. EF Core solves this in part by choosing what parts of a query can be translated to SQL, and then executing the rest of the query in memory. In some situations, this can be desirable, but in many other cases it can result in very inefficient queries that aren't identified until an application is in production.

In EF Core 3.0 we're planning to make deep changes to how our LINQ implementation works and how we test it, in order to make it more robust (for example, to avoid breaking queries in patch releases); to enable it to translate more expressions correctly into SQL; to have it generate efficient queries in more cases; and to prevent very inefficient queries from going undetected until production.

We've been working on a Cosmos DB provider for EF Core, to enable developers familiar with the EF programing model to easily target Azure Cosmos DB as an application database. The goal is to make some of the advantages of Cosmos DB—like global distribution, "always on" availability, elastic scalability and low latency—even more accessible to .NET developers. The provider will enable most EF Core features, like automatic change tracking, LINQ and value conversions, against the SQL API in Cosmos DB.

Other features we intend to include in EF Core 3.0 are property bag entities (entities that store data in indexed properties instead of regular properties); the ability to reverse-engineer database views into query types; and integration with new C# 8.0 features like IAsyncEnumerable<T> support and nullable reference types.

We understand that porting to EF Core can require a significant effort for many existing applications using previous versions of EF. For that reason, we're also porting EF 6 to work on .NET Core.

.NET Standard 2.1

When you adhere to the .NET Standard you can create libraries that work on all implementations of .NET, not only .NET Core but also Xamarin and Unity. In .NET Standard 1.x, we modeled only APIs that were already common across the various implementations. With .NET Standard 2.0, we focused on making it easier to port existing .NET Framework code to .NET Core, which resulted in not only an additional 20,000 APIs, but also compatibility mode, which enables you to reference .NET Framework libraries from .NET Standard-based libraries without having to recompile them. For both versions of the standard, there were almost no new components as all the APIs were existing .NET APIs.

For .NET Core 3.0 we're integrating the Blazor component model into ASP.NET Core. We call this integrated component model Razor Components.

With .NET Standard 2.1, this has changed: We've added about 3,000 APIs that are mostly brand-new and were introduced as part of the open source development of .NET Core. By adding them to the standard, we're bringing them to all implementations of .NET Standard.

Among these new APIs are:

- **Span<T>** In .NET Core 2.1 we added Span<T>, which is an array-like type that allows representing managed and

DevExpress®

From Desktops to Web and Mobile

Your Next Great App Starts Here

Experience the DevExpress difference and see why your peers consistently vote our products #1. With our Universal Subscription, you will build your best, see complex software with greater clarity, increase your productivity and create stunning applications for Windows, Web and your Mobile world.

DevExpress Universal ships with 500+ UI controls.
It also includes our royalty-free reporting and dashboard platform.

WIN ASP MVC WPF UWP JS

Download your free 30-day trial today.

devexpress.com/try

unmanaged memory in a uniform way and supports slicing without copying. Span<T> is at the heart of most performance-related improvements in .NET Core 2.1. Because it allows managing buffers in a more efficient way, it can help in reducing allocations and copying. If you want to learn more about this type, be sure to read Stephen Toub's excellent article on Span<T> (msdn. com/ magazine/mt814808).

C# 8.0 is the next version of C# and it improves the language in several major ways.

- **ValueTask and ValueTask<T>** In .NET Core 2.1, the most significant feature involved improvements in our fundamentals to support high-performance scenarios (bit.ly/2HflXob), which also included making async/await more efficient. ValueTask<T> already exists and allows you to return results if the operation completed synchronously without having to allocate a new Task<T>. With .NET Core 2.1, we've improved this further, making it useful to have a corresponding non-generic ValueTask that allows reducing allocations even for cases where the operation has to be completed asynchronously, a feature that types like Socket and NetworkStream now utilize.
- **General Goodness** Since .NET Core was open sourced, we've added many small features across the base class libraries, such as System.HashCode for combining hash codes or new overloads on System.String. There are about 800 new members in .NET Core and virtually all of them got added in .NET Standard 2.1.

For more details, check out the .NET Standard 2.1 announcement at bit.ly/2RCW2fX.

C# 8.0

C# 8.0 is the next version of C# and it improves the language in several major ways. Nullable reference types help prevent null reference exceptions and promote null-safe coding practices. You can opt in to the feature to get warnings when you assign null into variables or parameters of, for example, type string. If you *want* null, you have to say so by using a "string?" nullable reference type.

Async streams do for asynchronous streams of data what async/await did for single asynchronous results. A new framework type IAsyncEnumerable<T> is the async version of IEnumerable<T>, and can likewise be foreach'ed over and yield return'ed:

```
public static async IAsyncEnumerable<T> FilterAsync<T>(
  this IAsyncEnumerable<T> source,
  Func<T, Task<bool>> predicate)
{
  await foreach (T element in source)
  {
    if (await predicate(element)) yield return element;
  }
}
```

Among other features, default interface member implementations enable interfaces to add new members without breaking existing implementers. Switch expressions allow more concise pattern matching, and patterns can be recursive, digging deeper into tested values. For more details on C# 8.0 see aka.ms/csharp8.

How Will .NET Framework and .NET Core Move Forward?

.NET Framework is the implementation of .NET that's installed on more than 1 billion machines and thus needs to remain as compatible as possible. Because of this, it moves at a slower pace than .NET Core. Even security and bug fixes can cause breaks in applications because applications depend on the previous behavior. We'll make sure that .NET Framework always supports the latest networking protocols, security standards and Windows features.

.NET Core is the open source, cross-platform, and fast-moving version of .NET. Because of its side-by-side nature it can take changes that we can't risk applying back to .NET Framework. This means that .NET Core will get new APIs and language features over time that .NET Framework can't.

If you have existing .NET Framework applications, you shouldn't feel pressured to move to .NET Core if you don't need to take advantage of any of .NET Core's features. Both .NET Framework and .NET Core will be fully supported; .NET Framework will always be a part of Windows. Even inside of Microsoft we have many large product lines that are based on .NET Framework and will remain on .NET Framework. But moving forward, .NET Core and .NET Framework will contain somewhat different features.

.NET Core is the open source, cross-platform and fast-moving version of .NET.

Wrapping Up

.NET Core 3.0 is scheduled to release in the second half of 2019. It will have open source versions of WinForms and WPF for Windows desktop development. Entity Framework 6 will be included, as well. And ASP.NET Core, Entity Framework Core, .NET Standard and C# will all receive significant updates. This version of .NET Core should seriously be considered for new .NET applications. For more information, see aka.ms/netcore3preview1.

We're excited about the future of .NET and adding more workloads to .NET Core. I encourage you to try the preview of .NET Core 3.0 and send us feedback. ■

Scott Hunter *works for Microsoft as the director of Program Management for .NET overseeing the runtime, frameworks, managed languages (C#, F#, VB.NET) and .NET tooling. Before this Hunter was the CTO of several startups including Mustang Software and Starbase, where he focused on a variety of technologies—but programming the Web has always been his real passion.*

Thanks to the following Microsoft technical experts for reviewing this article: *Ankit Asthana, Damian Edwards, Richard Lander, Immo Landwerth, Beth Massi, Mads Torgersen*

File Format APIs

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

Try risk free - 30 day trial

ASPOSE

Download a Free Trial at

https://downloads.aspose.com

Aspose.Words

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

Aspose.Cells

Develop high performance .NET, Java and Android applications to Create, Edit or Convert Excel worksheets (XLS, XLSX, ODS etc).

Aspose.Pdf

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

Aspose.Slides

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

Aspose.Email

Create, Edit or Convert Outlook Email file formats (MSG, PST, EML etc.) and popular network protocols.

Aspose.BarCode

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

Aspose.Imaging

Deliver efficient applications to Create, Draw, Manipulate or Convert image file formats.

Aspose.Tasks

Develop high performance apps to Create, Edit or Convert Microsoft Project® document formats.

- Aspose.Diagram
- Aspose.Note
- Aspose.3D
- Aspose.CAD
- Aspose.HTML
- Aspose.GIS

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

NEW IN 2019!

On-Demand Session Recordings for One Year!

Get access to all sessions (not including Hands-On Labs or Workshops) at each show for a year. Learn more at vslive.com.

Developer Training Conferences and Events

Choose VSLive! For:

- ✔ In-depth developer training
- ✔ Unparalleled networking
- ✔ World-class speakers
- ✔ Exciting city adventures

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 An 1105 MEDIA company

JOIN US IN 2019!

February 6-7, 2019

Microtek Training Center Dallas

Want to learn more on ASP.NET Core in the Microsoft Cloud? Check out Visual Studio Live!'s upcoming Training Seminar to expand your knowledge and accelerate your career.

REGISTER NOW!
vslive.com/dallas

March 3-8, 2019

Bally's Hotel & Casino

Visual Studio Live! kicks off 2019 in the heart of Las Vegas with 6 days of hard-hitting Hands-On Labs, workshops, 60+ sessions, expert speakers and several networking opportunities included! Register to join us today!

REGISTER NOW!
vslive.com/lasvegas

April 22-26, 2019

Hyatt Regency

For the first time in our 20-year history, Visual Studio Live! is heading down south to New Orleans for intense developer training, bringing our hard-hitting sessions, well-known coding experts and unparalleled networking to the Big Easy!

REGISTER NOW!
vslive.com/neworleans

June 9-13, 2019

Hyatt Regency Cambridge

Join Visual Studio Live! for an amazing view of Beantown, bringin g our infamous speakers for intense developer training, Hands-On Labs, workshops, sessions and networking adventures to the Northeast.

REGISTER NOW!
vslive.com/boston

August 12-16, 2019

Microsoft HQ

Join our Visual Studio Live! experts at the Mothership for 5 days of developer training and special Microsoft perks unique to our other show locations. Plus, we are adding the ever-so popular full-day Hands-On Labs to the agenda in Redmond for the first time this year!

REGISTER NOW!
vslive.com/microsofthq

September 29, 2019

Westin Gas Lamp

Head to the heart of the San Diego Gaslamp District with Visual Studio Live! this Fall as we immerse ourselves with all things for developers, including several workshops, sessions and networking opportunities to choose from.

DETAILS COMING SOON!

October 6-10, 2019

Swissotel

Head to the Windy City and join Visual Studio Live! this October for 5 days of unbiased, developer training and bringing our well-known Hands-On Labs to the city for the first time.

DETAILS COMING SOON!

November 17-22, 2019

Royal Pacific Resort at Universal

Visual Studio Live! Orlando is a part of Live! 360, uniquely offering you 6 co-located conferences for one great price! Stay ahead of the current trends and advance your career – join us for our last conference of the year!

DETAILS COMING SOON!

What's New in Visual Studio 2019

Mads Kristensen

Visual Studio 2019 introduces exciting improvements and new features aimed at optimizing developer productivity and team collaboration. Whether you're using Visual Studio for the first time or have been using it for years, you'll benefit from features that improve all aspects of the development lifecycle—from smoother and more focused project creation to cloning from repository workflows, to driving the maintainability and quality of your code. Team and open source collaborative workflows are improved, as well.

One of the goals of the Visual Studio team was to make the upgrade to Visual Studio 2019 as seamless and simple as possible. So you'll find that there are no breaking changes in the format of solution and projects files when you step up, and that your existing code should open as expected.

There's more good news. The extensibility APIs in Visual Studio 2019 remain relatively unchanged, which means that any extension you use in Visual Studio 2017 can be updated with minimal effort to support Visual Studio 2019. Many extensions have been updated already, ensuring that they're available to you for the preview release. Updated extensions include Productivity Power Tools, Web Essentials, VsVim and many other popular extensions on the Visual Studio Marketplace today.

There are two other areas of particular interest to customers: performance and reliability. We've been shipping preview builds of new versions of Visual Studio for years, with updates more recently being released on a quarterly basis. These previews give you an early look at what's coming so you can work with the preview bits, and provide feedback to the Visual Studio product team. They also help you start thinking about how the new capabilities might impact you and your organization. Finally, these previews provide us with valuable feedback about issues in various user scenarios, as well as insight into how features might be tweaked to deliver the highest-quality product.

So, let's take a closer look at what you can expect to find in the preview of Visual Studio 2019.

This article discusses:

- New features and capabilities in Visual Studio 2019
- Updates to the Visual Studio Debugger
- UI enhancements and improved search functionality

Technologies discussed:

Visual Studio 2019, IntelliCode, CodeLens

Easier to Launch Your Code

The first thing you'll notice when you open Visual Studio 2019 is the new Start window, shown in **Figure 1**. It presents you with options to clone or check out code, open a project or solution, open a local folder, or create a new project. Whether you're new to Visual Studio or new to coding, or have been coding and using Visual Studio for years, these capabilities make getting to your code faster and more focused than ever.

We've concentrated on improving the time it takes to load solutions and projects, whether these are small or very large. Not only will solutions load faster, they'll do more of the work in the background, preventing extended UI delays and hangs during initialization.

It's worth noting that it's now possible to start a project by cloning or checking out from source control directly from the Start window. Bringing this feature front and center to the Start window finally makes it as simple as it should be.

When you create a new project, Visual Studio will open a redesigned project creation dialog. We received a lot of feedback about the old project dialog, so we optimized the experience to give you powerful search and filtering capabilities up front. This makes it easier to discover the right template for your new project. The same dialog is used on both the Start window and inside Visual Studio.

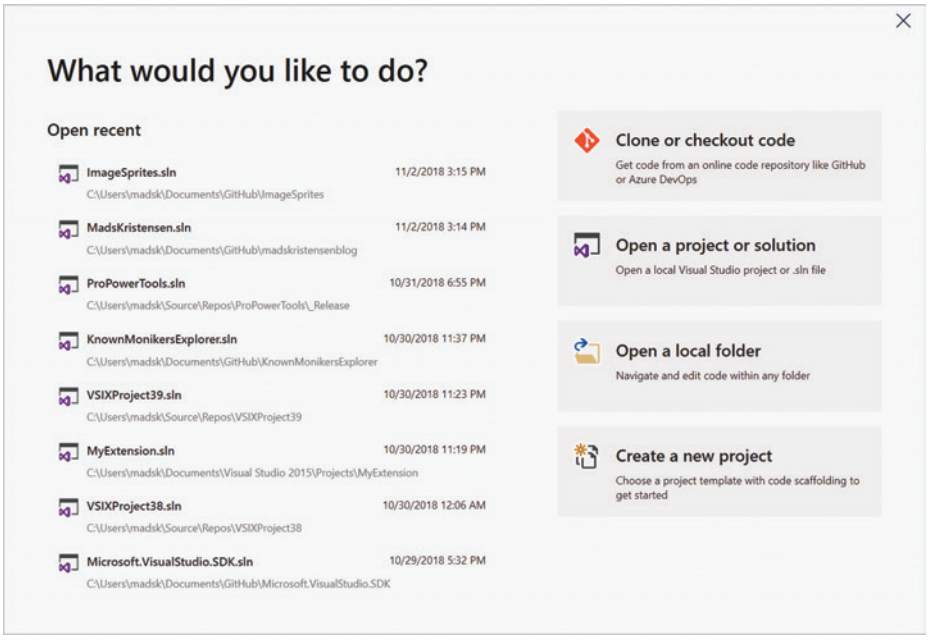


Figure 1 **The New Start Window**

Previous versions of Visual Studio made finding the right template for your app challenging, with the hundreds of templates shipped with Visual Studio, plus the thousands more made available for download by the community. The new dialog eases this task by bringing the most popular templates forward, while making it easy to browse templates and quickly filter them based on language, platform and project type.

Now you open the project creation dialog and the focus is directly in the search box, which filters the template list in real time as you type. The project creation dialog has always had a search box, but it never felt as natural, fast and convenient as it does in Visual Studio 2019.

More updates are coming to the project creation dialog, such as a list of your most recently used templates and other enhancements.

Simplified UI

To increase productivity, Visual Studio contains several updates to improve the UI and your experience, so you can focus on what matters—your code. When Visual Studio opens, you'll immediately notice some changes from previous versions. One of the most visible changes is the updated blue theme with softened edges around icons, toolbars and tool windows. This is the first major update to the blue theme since its introduction in Visual Studio 2012 and is a modernization of the interface. Similar updates to both the Light and Dark themes are planned for a future update.

Another visible change is the top-level menu, which has been moved up into the title bar, reclaiming vertical space to make room for more code in your editor. This change optimizes available space without changing how you navigate the IDE.

In future updates, expect to see additional subtle changes designed to simplify the UI while bringing focus to your projects and code documents.

A Better Search

The search feature, formerly known as Quick Launch, has been completely rewritten to make it faster and more accurate. It even provides a more forgiving search experience that can work with spelling mistakes.

Whether you're looking for commands, settings, installable components or other useful things, the new search feature makes it easier than ever to find what you're looking for within the IDE. Keyboard shortcuts are displayed next to search results for commands, so they can be more easily memorized for future use, as shown in **Figure 2**.

> The extensibility APIs in Visual Studio 2019 remain relatively unchanged, which means that any extension you use in Visual Studio 2017 can be updated with minimal effort to support Visual Studio 2019.

The new search helps speed things up by displaying results dynamically as you type in your query. It also accommodates for spelling mistakes and more natural language by using a fuzzy search algorithm. Finally, the results displayed have been refreshed to provide more relevant information, including any associated shortcuts for the keyboard-driven developer.

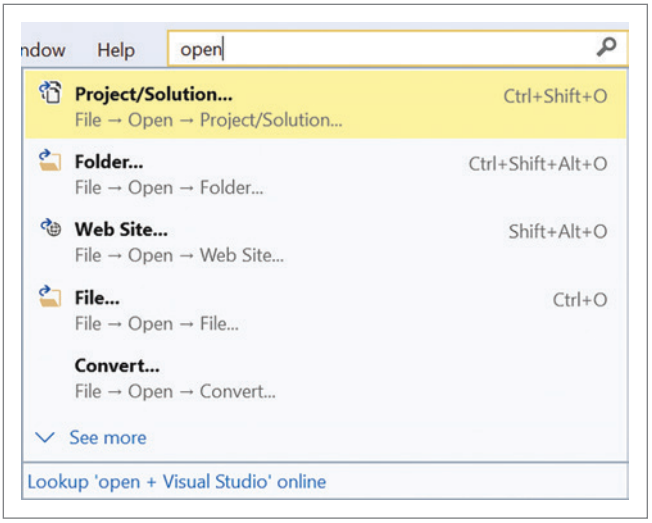


Figure 2 **The New Search Feature**

Initially, you can search across menus, commands, options and installable components. The Visual Studio team will continue to expand on search accuracy and incorporate other search providers to work toward providing a centralized search experience. You'll also notice that search has been moved up into the title bar, giving it a more prominent position that's easy to locate and out of the way when not being used.

AI-Assisted IntelliCode

IntelliCode is a set of AI-assisted capabilities that improve developer productivity, with features like contextual IntelliSense, code formatting and style rule inference. More improvements are in the works, including focused reviews for your pull requests in future updates. IntelliCode is an optional extension for both Visual Studio 2017 and 2019, and has received some major updates that make it even better. You can download the updated IntelliCode extension at aka.ms/vsintellicode. Read more about IntelliCode and its improvements in the post on The Visual Studio Blog at aka.ms/vsicblog.

Until now, IntelliCode recommendations have been based on learning patterns from thousands of open source GitHub repos. But what if you're using code that isn't in that set of repos? Perhaps you use a lot of internal utility and base class libraries, or domain-specific libraries that aren't commonly used in open source code, and you'd like to see IntelliCode recommendations for them, too. If you're using C#, you can have IntelliCode learn patterns from your code, so it can make recommendations for things that aren't in the open source domain.

When you open Visual Studio after installing the updated IntelliCode extension, you'll see a prompt that lets you know about training on your code, and will direct you to the brand new IntelliCode page to get started. You can also find the new page under View | Other Windows | IntelliCode. Once training is done, it will let you know about the top classes for which it found usage, so you can just open a C# file and start typing to try out the new recommendations. The trained models are kept secured, so only you and those who have been given your model's sharing link can access them. Your model and what it learns about your code stay private to you.

Many developers have requested IntelliCode recommendations for their favorite languages. With this update, Microsoft is excited to add four more languages that can get AI-assisted IntelliSense recommendations. C++ and XAML have been added in the extension for Visual Studio, while TypeScript/JavaScript and Java have been added to Visual Studio Code.

CodeLens for Everyone

CodeLens has been a feature found only in Visual Studio Enterprise, but that will change in an upcoming preview of Visual Studio 2019, when it will also be available for the Community edition, likely in 2019. CodeLens shows the number of references a type or method has, information about unit tests covering the method, and data directly from Application Insights.

In addition, Microsoft has made CodeLens fully extensible, so third-party extensions can start to add their own experiences on top of it. CodeLens makes key information about your types easy to find, while keeping you in the source code. Lenses for source control history and IntelliTrace are still an Enterprise-only feature.

> The project creation dialog has always had a search box, but it never felt as natural, fast and convenient as it does in Visual Studio 2019.

Code Cleanup

It can sometimes be easy to overlook an error, warning or suggestion in your code, because it has scrolled out of view or the Error List window isn't visible. Those issues often aren't caught until the next time the project is built. On top of that, warnings and suggestions based on code style rules from .editorconfig files or code analyzers can go unnoticed and might not get caught until it's time for a code review before committing the code changes to source control.

Situations like these could be avoided if there were an always-visible indicator showing when the code file contains errors, warnings and suggestions. It would be even better if the available code fixes provided to take care of the warnings and suggestions could be applied to the whole code file—or even the project—in one fell swoop.

That's why Visual Studio 2019 has added a document health indicator icon at the bottom right side of the editor (it's likely to be relocated in a future update). The icon is red, yellow or green, indicating the health level of the code file or document. **Figure 3** shows how this looks.

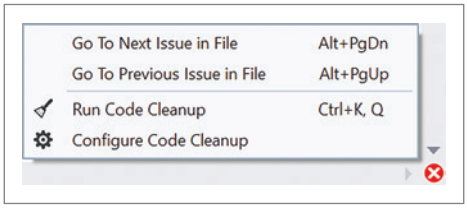


Figure 3 **The Document Health Indicator**

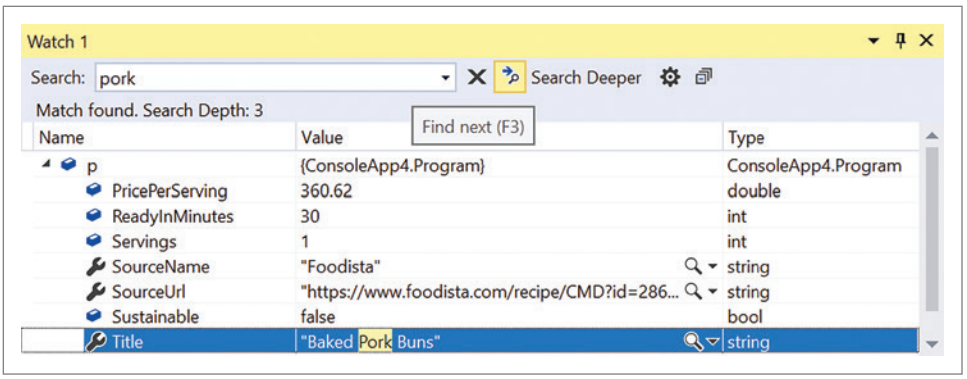


Figure 4 **Search Within a Watch Window**

Right-clicking the indicator icon will show shortcuts for dealing with any issues in the document, as well as run a full document-wide cleanup process. The cleanup will format the code and apply any code fixes suggested by the current settings, .editorconfig files or Roslyn analyzers.

This is a huge productivity boost designed to help you write more maintainable code faster, and to deal with issues earlier, before they enter a code review. It's important to note that this feature is undergoing some UI work for future updates, but the Visual Studio team decided to ship this early UI already in Preview 1 to maximize your productivity and hopefully get feedback from you to make this capability even better.

Debugger Updates

Visual Studio 2019 aims to improve your productivity in everyday tasks, reducing everyday friction. One of the new features helping reduce that friction is the ability to search in the various Watch windows (locals, auto and watch during debugging). You've probably been there before, looking in the Watch window for a string among a sea of values. Now, Visual Studio can do that looking for you!

> To increase productivity, Visual Studio contains several updates to improve the UI and your experience, so you can focus on what matters—your code.

In **Figure 4** you can see a search for "pork," which searches within the watch names, values and type columns. When you search, the software highlights any visible text matches and searches, based on the depth of the search, through the window. You can navigate through all found matches by using the Find next (F3) command. One of the challenges with searching a watch is that the data represented can be recursive, or you simply have a very deep parent chain. To get around this limitation, by default, we only search three levels deep. If you don't find something, you can just use the Search Deeper button to search through the next two levels. You can also change the default search depth in Tools | Options.

So, for the first time, you can now easily search through arrays of any kind to find just the item you're looking for. This has traditionally been a pain point where you manually had to look at each item in the array to pinpoint the one you need. Now it's as simple as typing in the search string.

We've done a lot in this release to improve performance. The Watch window and the other debugging tool windows such as Locals and Call stack have all undergone a redesign that allows them to load asynchronously. Now many of the processes that ran on the UI thread in Visual Studio run on a non-blocking background thread. The result is a clear improvement in performance and fewer UI delays or hangs while debugging.

Debugging has gotten better, too. In addition to improving the responsiveness of the debugger, we're continuing the work started in Visual Studio 2017 to move memory-intensive components and workloads into separate processes. With Visual Studio 2019, components of the C++ debugger will run in a separate process, which means large memory-hungry programs can be debugged without Visual Studio running out of memory. And step debugging now provides faster and smoother step-through code, making it the best overall debugging experience yet.

A lot more improvements are coming to the debugger in future releases, both in the terms of new features and performance improvements.

Wrapping Up

In addition to the features mentioned in this article, there are powerful new team collaboration capabilities that promise to revolutionize developer interaction. Read more in Julie Lerman's article, "Remote Collaboration with Visual Studio Live Share," in this issue.

The features described in this article are just a sampling of what's available in Visual Studio 2019 Preview 1. There are plenty more fixes, tweaks and additional functionality beyond what you've read about, including .NET Core 3 preview tooling, mobile development, cross-platform C++ and Azure features.

The Visual Studio 2019 Preview was not yet feature complete at the time of this writing, but it clearly shows the direction you can expect the final version of the product to take. If you haven't already, now's a good time to head over to visualstudio.com to download the preview and try out these new features for yourself. ■

Mads Kristensen *is a senior program manager on the Visual Studio Extensibility team. He is passionate about extension authoring, and over the years he's written some of the most popular extensions with millions of downloads.*

Thanks to the following Microsoft technical experts for reviewing this article: *Gordon Hogenson, Rajen Kishna*

Collaborative Development with Visual Studio Live Share

Julie Lerman

I hope I'm not embarrassing myself to express how excited I am by Visual Studio Live Share! The first time I saw a demonstration of an early preview, I immediately found an excuse to use it in a live streaming session where Jeff Fritz and I worked on a .NET Core project together, each on our own computer, nearly 400 miles apart.

Live sharing is not a screen-sharing session, of the sort I use to help non-techie friends when they're confounded by some software on their computer. Jeff was coding on his computer in Visual Studio in Windows while I worked on the same code on my MacBook in Visual Studio Code. What he typed, deleted or debugged, showed up in my code. Whatever I did to affect the code appeared in Visual Studio on his machine. When he opened a file in Visual Studio, that file opened up in Visual Studio Code on my machine. Yet we had our own cursors, and could work both together and independently.

Sounds magical? I thought so, too. So I have really dug in further with my usual battery of "what-if" questions and learned so much more about this incredible new tool. Let's take a look.

Visual Studio Live Share is currently in preview. All information is subject to change.

This article discusses:

- Starting a Live Share collaboration session
- Co-editing and co-debugging with other developers
- Sharing localhost servers and terminals
- Best practices for securely sharing your code

Technologies discussed:

Visual Studio Live Share, Visual Studio, Visual Studio Code

Under the Covers

Once you've authenticated for a session using either your Microsoft or GitHub credentials, Live Share attempts to connect with other developers in a peer-to-peer fashion, but will fall back to an Azure relay if necessary. When collaborating with an external participant, this happens through Azure and it does mean you need to be on the Internet. To provide guests with full access to your solution, without actually requiring you to upload any of your code, Live Share communicates only the file system structure of your project to others. When someone opens a file, it sends the file contents. As changes are being made, it sends the character differences. When debugging, it sends debug steps and debug state. Shared testing is coming in the near future. Live Share collects that data coming in and displays it in your IDE. It can respond to a collaborator opening files and automatically open that

Figure 1 **The Live Share Button Lets You Activate a New Live Share Session**

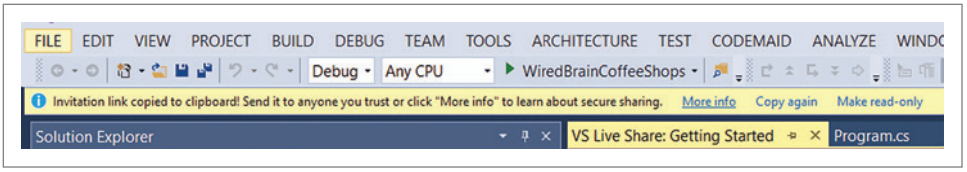


Figure 2 **The Information Bar Displayed upon Starting a Live Share Session**

But enough teasing—let's take a look at how to work with Live Share. I'm going to demonstrate using multiple instances of Visual Studio on a single computer so you can follow along if you'd like without having to find a collaborator right away. And it's a great way to practice and explore the features on your own.

file in another's IDE. But because you may want to edit different files at the same time, you can disable that automatic file switching.

Collaboration Game Changer

This experience is so many times better than having a client show me their code on a screen-share session in Skype. I live on a mountainside with limited Internet speed, and the fact that Live Share transmits only text file contents and minimal data when edits are being made means sharing can be much faster than transmitting entire screen images across the Internet. Moreover, when working with others over Skype, I have to constantly ask, "Can you click on this," or, "Scroll to that," or, "What happens if you change that code on line … oh wait, can you open settings and get line numbers to show?" Even if I use a tool that allows me to control their mouse, such as LogMeIn, it's cumbersome. There are always latency issues, and working in someone else's IDE is not always fun. They often have things set up differently. Sometimes their color schemes make it hard for me to see. Or their fonts are too small. All of these things get in the way of smooth, effective collaboration. This has been my experience for years collaborating with or coaching developers and teams without having to get on a plane. I'm sure for any of you with distributed teams, it all sounds much too familiar.

> Live Share is currently available as an extension for Visual Studio 2017, although it will be a built-in feature of Visual Studio 2019.

This is why I'm so excited by what Live Share can do. Whether I'm doing code reviews, helping someone solve a "works on my machine but not theirs" problem or doing pair programming with someone, Live Share is an enormous game changer. And it's not just me having this reaction. You might find the #vsliveshare hashtag on twitter to be entertaining as you scroll through tweets by developers as they are first discovering it.

Live Share can be used with a wide variety of languages and platforms. I've even used it with an old .NET 4 WinForms solution that uses VB.NET! The language support is broader for VS Code than Visual Studio, but in Visual Studio you can use C#, VB.NET, F#, C++, Python and more. Check out the Language and Platform Support page at bit.ly/20qjds3.

Figure 3 **Visual Studio Showing That It Has Joined a Live Share Session with JL**

Activating a Live Share Session

Live Share is currently available as an extension for Visual Studio Code and Visual Studio 2017, although it will be a built-in feature of Visual Studio 2019. As an extension, it's still called a preview. I'll focus on the experience in Visual Studio 2017.

Live Share is, by default, associated with your Visual Studio login and the extension adds indicators next to your login information and avatar in the UI (see **Figure 1**). The Live Share button lets you activate a session and requires that you already have your target solution, project or folder open in Visual Studio. You can also initiate a session from the File menu. Once you click it, the text will change from "Live Share" to "Sharing."

The first time you activate Live Share in Visual Studio, it will alert you to the fact that it's defaulting to collaborating through a local network connection and that you'll be prompted to allow relevant firewall settings. If you choose to deny opening those ports, it will change its behavior to route all of the connections—even those on the same network—through Azure instead. This is something you can change as needed down the road.

Why would you need it for a local network connection? It turns out that Live Share is already being widely adopted by teams that are co-located. Imagine being able to help someone on your team with a quick problem with only a minor interruption to your workflow. When pair programming with someone, you're usually both hovering over one keyboard and one screen, and elbow room can be an issue. When teams are mob programming, everyone's looking at a projected screen from one computer. Instead, you could sit side by side (or across the hall or on another floor) and still work together but with each developer looking at an IDE set up the way they like, allowing each user to truly participate. When it's time to switch hands on a keyboard, although the action of physically moving into "driver" position has a lot of merits, it's a lot easier and again, nobody has to work on an unfamiliar machine with an environment that might be awkward for them.

After initiating the session, the very next thing you'll see is a yellow bar below the menu, as shown in **Figure 2**, telling you that the invitation link has been copied to your clipboard so you can share with any guests. This is a good reminder that the person whose code is to be worked on is the one who needs to initiate the session. You can't switch directions in the same way you may be used to with screen sharing.

A "More info" link opens a window with some Getting Started information about Live Share.

Amazingly, you can invite up to five people to your session. However, this can be potentially chaotic, but you can control who can interact with the code and who can only view what's happening. A separate link makes the collaboration session read-only to a guest.

The invitation link is a URL that you can either paste into a browser or directly into Visual Studio under the File | Join Collaboration session menu option. If you're a guest using a browser, you'll be brought to a Web page showing the identity of the user that created the invitation and a pop-up window prompting you to open the "Pick an app" Windows interface. From there you can select either Visual Studio or Visual Studio Code, and the chosen IDE will open and set up the Live Share session. As a guest, you'll see indicators next to your account showing that you've joined a session with, in this case, "JL" (see **Figure 3**). The solution, project or folder shared through the link will be open in Solution Explorer. In my Solution Explorer, I can see that the code that was relayed to my machine is temporarily stored in a folder inside C:\Users\Julie\AppData\Local\Temp\.

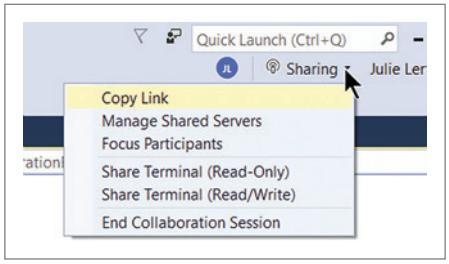


Figure 4 **The Sharing Menu Dropdown with Host-Specific Options**

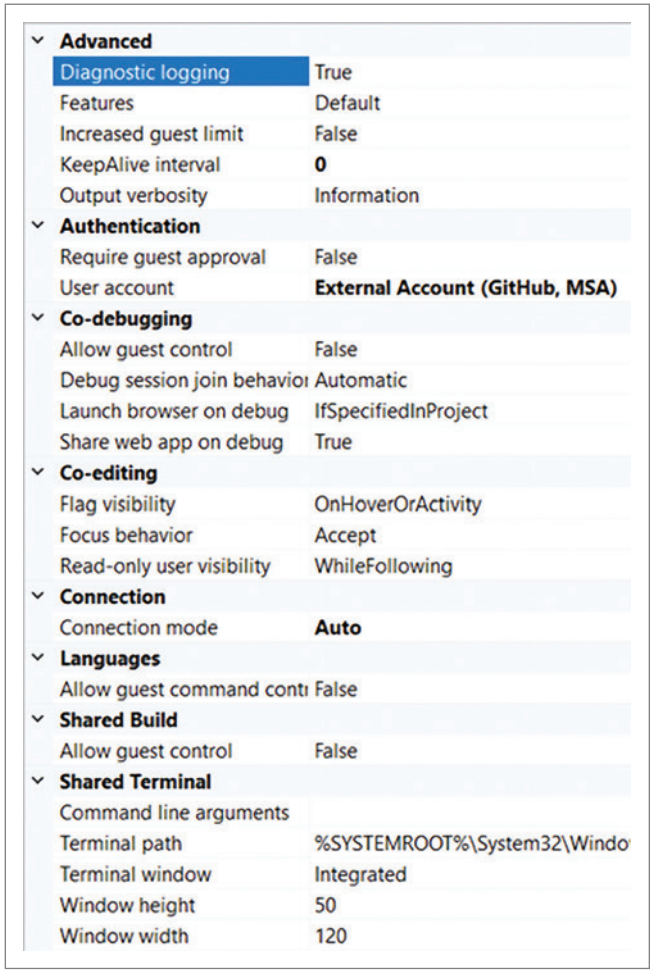


Figure 5 **Controlling Live Share Behavior Using Settings in Tools | Options**

Note that only the file structure and code files have come across. When I look in the temp folder, I see no .csproj or .sln files. And building leaves no bin or debug in the temp folder. Even if you change the defaults and allow guests to trigger a build, that build happens on the host. It's all about displaying the files on the guest, not recreating the solution or projects.

On the Web page that opened from the invitation link, there are also links for guests to install the extension to either of these IDEs if they don't already have it. I like that it's this simple and discoverable, so you don't need to send lots of instructions along with the link.

And as mentioned earlier, you don't need to use the same OS or IDE that initiates the session. I have both Visual Studio 2017 and VS Code on my Windows machine. I'll use Visual Studio 2017 as an invited collaborator. Notice that I'm playing both host and guest. You can do this with multiple instances of Visual Studio or VS Code on the same machine, which is a great way to get familiar with Live Share. Note that currently there's no Live Share extension for Visual Studio for Mac.

Once connected, a guest machine will have some bling added to its UI, including an icon, referred to as a badge, with the initials of the host. That's the blue circle in **Figure 3**. (I'll explain the ring around the circle shortly.) The Live Share button indicates the guest has Joined a session. The Live Share button on my host instance of Visual Studio has a "Sharing" dropdown similar to the "Joined" dropdown in **Figure 3**. It also shows a prompt telling me that a guest has joined the session and I can accept or reject this connection. I've clicked OK to accept it. When using two instances of Visual Studio on one computer, you can't have two different Visual Studio login identities, so I'll be Julie Lerman with a blue JL badge in both instances.

By default, the guest will follow the host wherever the host wanders in the solution. So, if I open up the program.cs file in the host instance, that file opens in the guest's IDE. But if the guest opens up a different file, that file won't automatically open on any other participant's IDE. This behavior is controlled by actions in the Live Share button dropdowns, as well as the badge. The dropdowns—Joined or Sharing (**Figure 4**) both have a "Focus Participants" option. When that's selected, the other participants will be notified that they'll now follow the focus of that collaborator. Notice the extra ring around the blue badge in **Figure 3**. This indicates that this IDE is following the other collaborator. You can click the badge to toggle the focus on and off. This is super handy when your collaboration changes from, "Let's work together on the same file," to, "Oh, wait, can you go check out a different file to copy some text?"

But the dropdown and badge provide only session-specific actions. The extension also adds a few dozen settings in Tools | Options in a section called Live Share, as shown in **Figure 5**.

Another interesting aspect of the guest IDE is that if you right-click on projects or files in Solution Explorer, you'll see only one item on the context menu—"Go to Git Changes." In my case, this was

DocuVieware

Universal HTML5 and Document Management Kit

Easy integration

Full support for custom snap-in

Zero-footprint solution

Fully customizable UI

Mobile devices optimization

Fast & crystal-clear rendering

Check the New Features and the Online Demos
60-day Free Trial Support Included at **www.docuvieware.com**

Powered by **GdPicture**.NET

DocuVieware is an ORPALIS imaging technologies product

```
20          }
21          Julie Lerman
22          private static void StoreAndRetrieveALocation()
23          {
24              throw new NotImplementedException();
25          }
```

Figure 6 **A Notation Appeared When a Participant Moved Their Cursor to the Beginning of This Line**

```
22          private static void StoreAndRetrieveALocation()
23          {
24              throw new NotImplementedException();
25          }
26          Julie Lerman
```

Figure 7 **All Participants Can See When Julie Lerman Selects Text**

initially disabled. Right-clicking the solution in Solution Explorer reveals a menu with options to add a file or folder, as well as modify, debug and launch settings. The reason for the limited context menus is, again, because the solution and project files aren't available. So Visual Studio treats these as simple files, not .NET projects.

Co-Editing with Your Collaborators

Anyone who has default (as opposed to read-only) access can edit files. A host can add or rename folders and files in Solution Explorer.

The default editing behavior, whether by host or guest, begins with mouse clicks to select text or a location in code. When a participant moves their cursor, the other participants will see a marker in their text. For example, in **Figure 6**, the guest, Julie Lerman, notices a method beginning on line 22 that isn't even implemented and recommends refactoring it out. Host Julie says, "Go for it!" The guest begins by clicking at the start of that line, and a marker shows up in the other participants' IDEs, as well as the name of the user who performed the action. The name will fade away, but the marker remains.

As the guest selects the four lines of the method, the other users can see the selection expand in their own IDEs, along with her name (see **Figure 7**).

And when she hits the delete key, the lines are also cut from the other participants' code.

When the host is using source control, the change also appears as a change in Team Explorer.

That's really the meat of co-editing. So, let's move on to another fun feature—co-debugging.

Co-Debugging

Earlier I mentioned that debugging steps and breakpoints are captured and transmitted by Live Share. This is another cool feature to see in action.

When one participant sets a breakpoint, that breakpoint is automatically set across all the participant IDEs. If the host changes the default setting (see **Figure 5**) to allow guests to debug, guests can also trigger debugging on the host. Then each IDE will run and stop at the breakpoints displaying available details. However, viewing those details is up to each guest. One might have the errors window open; another might prefer to hover over the breakpoint code to view debug information.

This brings up the question of building projects or solutions. I mentioned earlier that, by default, guests aren't allowed to build. In fact, the Visual Studio BUILD menu is removed from guest IDEs during a sharing session, and to keep things secure by default, only the host can build. In this case, the guest will display build output, but on careful inspection, I noticed that the output was coming from the host; that is, the build output was displaying the file paths of the host. That's because the build is happening on the host. If you revisit **Figure 5**, you'll see there's an option to allow guests to build. But if you enable that setting, you'll need to begin a new session for it to kick in on the guest machines. Debugging is the same. By default, guests can't trigger a debug session, but you can change that setting on the host. Note that guests can't skirt this limitation and use command-line interface (CLI) commands to build or debug. Remember, the projects are not on their computers. In a bit I'll tell you about using the Shared Terminal feature to share the host's terminal, which allows users to see and run command-line commands on the host.

Debug actions, such as step or skip-over, are also relayed to collaborators. As you step through your code, they're following along, even if they've disabled follow mode (by clicking on the collaborator badge to remove the ring around it, as shown in **Figure 3**).

Debugging can mean more than just stepping through code. If you're working in an ASP.NET Core project such as a Web site or a Web API, you can share the server used by the host so that guests can physically browse across the network or Internet to the host's server. I was a little confused by this at first because it seems a little scary to think that my dev machine's server would be accessible on the Internet. But the sharing is done using a secure SSH or SSL tunnel. By default (and configurable), the server will be shared automatically when you start debugging (F5) or run (Shift F5). If you prefer to explicitly share servers, you can do so through the Manage Shared Servers option on the host's Sharing dropdown, as shown in **Figure 8**.

It's also possible to explicitly share other servers hosted on TCP. For example, if you have a SQL Server available on TCP, you could expose its port and let your collaborators connect to that database. Read more about shared servers, as well as some precautions around this feature at bit.ly/2FlWnlU.

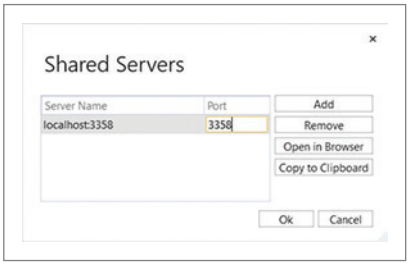


Figure 8 **Setting Up a Shared Server**

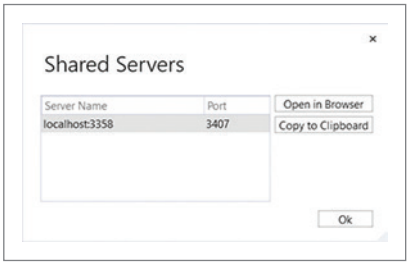


Figure 9 **Exposing an Available Port for Sharing Servers**

Once the servers are shared and the application is running on the host, guests can use their Live Share dropdown list to select the View Shared Servers

option. This displays a list of servers the host has shared. **Figure 9** shows an interesting feature in action: Port 3358 is already in use on my machine, so Live Share has enlisted an available port (3407) to expose 3358 through on my machine.

As a guest, I chose Open in Browser and thought at first that something was wrong because I got a 404 error: "No webpage was found for the web address: http://localhost:3407/." But that's because I was running a Web API and needed to navigate to the proper URL—"http://localhost:3407/api/values." Once I did, there was the output from the API in my guest's browser.

If you're debugging a Console application, note that VS Code can open up the console window in a guest session, but Visual Studio 2017 is not yet able to do that.

Shared Terminals, Too!

If you're building .NET Core apps in Visual Studio 2017, you may not be relying a lot on the .NET CLI or other command-line tools. VS Code users are much more likely to be using the CLI, as they don't have all of the bells and whistles Visual Studio brings to the table. However, if you are running terminal commands in the context of writing, running or debugging your app, you might also want to use the shared terminal feature to share the console window among participants.

The host can share a read-only or read/write terminal from the Sharing dropdown. This immediately opens a terminal window on both host and guest machines. Note that if you've installed the Whack Whack Terminal extension (bit.ly/2PuVDzu), which enables an integrated terminal in Visual Studio 2017, Live Share will use that instead. As the host types, the guests can see each keystroke hitting their terminal window. If the guests have write access, anything they type will show on the other participants' terminals. If a guest runs, for example, the "dotnet build" command, the build will take place, although just like building from Visual Studio, that happens on the host's machine.

Yes, It's Secure!

I've already noted a number of behaviors related to security. Hosts and guests are validated using either their Microsoft or GitHub credentials. Sessions are shared by default as editable, but there's a read-only invitation link to share, as well. When guests join a session, the host is notified and can accept or reject that guest, as well as kick a guest off any time. By default, guests can't trigger builds or debugging, and project and solution files aren't even available on a guest computer.

There are other security features that the host can control. You can:

- Delay guests connecting to a shared session through the "Require guest approval" setting under Authentication, as shown in **Figure 5**. Think of this as a Skype lobby.
- Control file access and visibility to a folder or project using a file named .vsls.json, where you can specify which files should be limited.
- Control guest access even more strictly in an Active Directory environment

These are all important considerations when employing Live Share to collaborate in your small or large enterprise teams. You'll find more details about these features in the Security Features of Live Share document at bit.ly/2AMyobo.

Ending a Session

The host can end a session from the Sharing dropdown and all of the guests will be kicked out of the session (with a friendly notification). The temp files will also get cleaned up on the guest machines. Of course, guests can leave a session at any time and the host will get a notification when that happens.

Interestingly, my Internet connection dropped during the session (something odd with my router). Live Share responded with a notification in the host system that the relay listener had gone offline and that I should consider resharing when it came back online.

Live Share Has a Big Future!

Visual Studio Live Share is a "sea change" feature for developers using Visual Studio and Visual Studio Code for a wide array of use cases. Distributed teams are becoming more and more common, so the ability of team members to work together on code in this way is extraordinary. But the benefits are not only for distributed teams, and there are so many other use cases—mentoring, code reviews, teaching, helping with problems. I can't tell you how many times someone has asked me about a strange problem they were having with their code and I responded that I really couldn't tell off the top of my head and would need to watch them debug through it myself. Live Share can make this so easy to do now. The docs even have a page with all types of use case ideas for you (aka.ms/vsls-usecases). Some I hadn't even thought of myself, such as coding interviews. This beats a whiteboard interview every time—and can also be done remotely!

A final thought—compared to sharing screens over Skype, the minimal data transfer can offer significant benefits for developers with slow or undependable Internet connections. My own Internet speeds are limiting because I live in a rural location in the United States. But this is a small inconvenience: usually means turning off the camera when I'm using Skype for voice and screensharing. But I've also mentored some developer friends in Lagos, Nigeria (an amazing tech hub), and other parts of the world, where the Internet connection is intermittent and, if I recall correctly, they used their cell phones to relay Internet access to their computers. For me, that really nails it—that not only does Live Share bring enhanced productivity to developers in general, it can also be an amazing game changer for developing communities. ■

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

Thanks to the following Microsoft technical expert for reviewing this article: *Jonathan Carter*

ML.NET: The Machine Learning Framework for .NET Developers

James McCaffrey

The ML.NET library is a new open source collection of machine learning (ML) code that can be used to create powerful prediction systems. Many ML libraries are written in C++ with a Python API for easier programming. Examples include scikit-learn, TensorFlow, CNTK and PyTorch. However, if you use a Python-based ML library to create a prediction model, it's not so easy for a .NET application to use the trained model. Fortunately, the ML.NET library can be used directly in .NET applications. And because ML.NET can run on .NET Core, you can create predictive systems for macOS and Linux, too.

The ML.NET library is still in preview. All information is subject to change.

This article discusses:

- The demo program and data
- Creating and training the model
- Saving and evaluating the model
- Using the trained model
- The new ML.NET API approach

Technologies discussed:

ML.NET, Visual Studio 2017, .NET Framework, .NET Core

Code download available at:

msdn.com/magazine/1318magcode

A good way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo creates an ML model that predicts the annual income for a person based on their age, sex and political leaning (conservative, moderate, liberal). Because the goal is to predict a numeric value, this is an example of a regression problem. If the goal had been to predict political leaning from age, sex and income, it would be a classification problem.

The demo uses a set of dummy training data with 30 items. After the model was trained, it was applied to the source data, and achieved a root mean squared error of 1.2630. This error value is difficult to interpret by itself and regression error is best used to compare different models.

The demo concludes by using the trained model to predict the annual income for a 40-year-old male with a conservative political leaning. The predicted income is $72,401.38. The demo in **Figure 1** was written using the ML.NET legacy approach, which is a good way for beginners to get a feel for ML.NET. In the second half of this introductory article, I'll discuss a newer approach that's somewhat more difficult to grasp but is the better approach for new development.

This article assumes you have intermediate or better programming skill with C#, but doesn't assume you know anything about the ML.NET library. The complete code and data for the demo program are presented in this article and are also available in the accompanying file download. As I'm writing this article, the ML.NET library is still in preview mode and is being developed

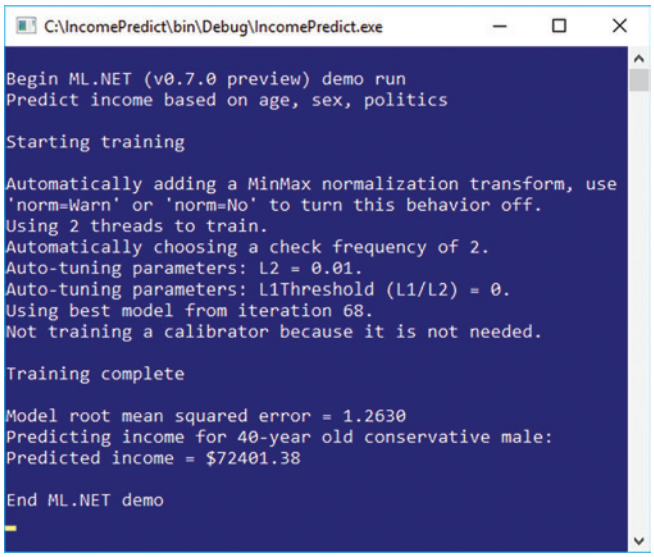


Figure 1 **ML.NET Legacy Demo Program in Action**

very quickly, so some of the information presented here may have changed a bit by the time you're reading this.

The Demo Program

To create the demo program, I launched Visual Studio 2017. The ML.NET library will work with either the free Community Edition or one of the commercial editions of Visual Studio 2017. The ML.NET documentation states that Visual Studio 2017 is required, and in fact I couldn't get the demo program to work with Visual Studio 2015. I created a new C# console application project and named it IncomePredict. The ML.NET library will work with either a classic .NET Framework or a .NET Core application type.

After the template code loaded, I right-clicked on file Program.cs in the Solution Explorer window and renamed the file to IncomeProgram.cs, and I allowed Visual Studio to automatically rename class Program for me. Next, in the Solution Explorer window, I right-clicked on the IncomePredict project and selected the Manage NuGet Packages option. In the NuGet window, I selected the Browse tab and then entered "ML.NET" in the Search field. The ML.NET library is housed in the Microsoft.ML package. I selected the latest version (0.7.0) and clicked the Install button. After a few seconds Visual Studio responded with a "successfully installed Microsoft.ML 0.7.0 to IncomePredict" message.

At this point I did a Build | Rebuild Solution and got a "supports only x64 architectures" error message. In the Solution Explorer window, I right-clicked on the IncomePredict project, and selected the Properties entry. In the Properties window, I selected the Build tab on the left, then changed the Platform Target entry from "Any CPU" to "x64." I also made sure that I was targeting the 4.7 version of the .NET Framework. With earlier versions of the Framework I got an error related to one of the math library dependencies. I again did a Build | Rebuild Solution and was successful. When working with preview-mode libraries such as ML.NET, you should expect quite a few hiccups like this.

The Demo Data

After creating the skeleton of the demo program, the next step is to create the training data file. The data is presented in **Figure 2**. If you're following along, in the Solution Explorer window, right-click on the IncomePredict project and select Add | New Folder and name the folder "Data." Placing your data in a folder named Data isn't required but it's a standard practice. Right-click on the Data folder and select Add | New Item. From the new item dialog window, select the Text File type and name it PeopleData.txt.

> The ML.NET library will work with either the free Community Edition or one of the commercial editions of Visual Studio 2017.

Copy the data from **Figure 2** and paste it into the editor window, being careful not to have any extra trailing blank lines.

The 30-item dataset is artificial. The first column is a person's age. The second column indicates sex and is pre-encoded as male = -1 and female = +1. The ML.NET library has methods to encode text data, so the data could've used "male" and "female." The third column is annual income to predict, with values divided by 10,000. The last column specifies the political leaning (conservative, moderate, liberal).

Because the data has three predictor variables (age, sex, politic), it's not possible to display it in a two-dimensional graph. But you can get a good idea of the structure of the data by examining the graph of just age and annual income in **Figure 3**. The graph shows that age by itself can't be used to get an accurate prediction of income.

After you create the training data in a Data folder, you should create a folder named Models to hold the saved model because the demo code assumes a Models folder exists.

Figure 2 **People Data**

```
48, +1, 4.40, liberal
60, -1, 7.89, conservative
25, -1, 5.48, moderate
66, -1, 3.41, liberal
40, +1, 8.05, conservative
44, +1, 4.56, liberal
80, -1, 5.91, liberal
52, -1, 6.69, conservative
56, -1, 4.01, moderate
55, -1, 4.48, liberal
72, +1, 5.97, conservative
57, -1, 6.71, conservative
50, -1, 6.40, liberal
80, -1, 6.67, moderate
69, +1, 5.79, liberal
39, -1, 9.42, conservative
68, -1, 7.61, moderate
47, +1, 3.24, conservative
18, +1, 4.29, liberal
79, +1, 7.44, conservative
44, -1, 2.55, liberal
52, +1, 4.71, moderate
55, +1, 5.56, liberal
76, -1, 7.80, conservative
32, -1, 5.94, liberal
46, +1, 5.52, moderate
48, -1, 7.25, conservative
58, +1, 5.71, conservative
44, +1, 2.52, liberal
68, -1, 8.38, conservative
```

The Program Code

The complete demo code, with a few minor edits to save space, is presented in **Figure 4**. After the template code loaded into Visual Studio, at the top of the Editor window I removed all namespace references and replaced them with the ones shown in the code listing. The various Microsoft.ML namespaces house all ML.NET functionality. The Threading.Tasks namespace is needed to save or load a trained ML.NET legacy model to file.

Notice that most of the namespaces have a "Legacy" identifier. The demo program uses what's called the

pipeline API, which is simple and effective. The ML.NET team is adding a new, more flexible API, which I'll discuss shortly.

The program defines a nested class named IncomeData that describes the internal structure of the training data. For example, the first column is:

```
[Column("0")]
public float Age;
```

Notice that the age field is declared type float rather than type double. In most ML systems, type float is the default numeric type because the increase in precision you get from using type double is rarely worth the resulting memory and performance penalty. Predictor field names can be specified using the ColumnName attribute. They're optional and can be whatever you like, for example [ColumnName("Age")].

The demo program defines a nested class named IncomePrediction to hold model predictions:

```
public class IncomePrediction {
  [ColumnName("Score")]
  public float Income;
}
```

The column name "Score" is required but, as shown, the associated string variable identifier doesn't have to match.

Creating and Training the Model

The demo program sets up an untrained ML model using these statements:

```
var pipeline = new LearningPipeline();
string dataPath = "..\\..\\Data\\IncomeData.txt";
pipeline.Add(new TextLoader(dataPath).
  CreateFrom<IncomeData>(separator: ','));
```

You can think of a LearningPipeline object as a meta container that holds the training data and a training algorithm. This paradigm is quite a bit different from those used by other ML libraries. Next, the pipeline performs some data manipulation:

```
pipeline.Add(new ColumnCopier(("Income", "Label")));
pipeline.Add(new CategoricalOneHotVectorizer("Politic"));
pipeline.Add(new ColumnConcatenator("Features", "Age",
  "Sex", "Politic"));
```

The legacy version of ML.NET requires that the column that holds the values to predict be identified as "Label" so the ColumnCopier method creates an in-memory duplicate of the Income column. An alternative approach is to simply name the Income column as Label in the class definition that defines the structure of the training data.

The trainer only works with numeric data so the text values of the Politic column must be converted to integers. The CategoricalOneHotVectorizer method converts "conservative," "moderate," and "liberal" to (1, 0, 0), (0, 1, 0), and (0, 0, 1). An alternative approach is to manually pre-encode text data.

The ColumnConcatenator method combines the three predictor columns into a single column named Features. This naming scheme is required. The training algorithm is added to the pipeline, and the model is trained like so:

```
var sdcar = new StochasticDualCoordinateAscentRegressor();
sdcar.MaxIterations = 1000;
sdcar.NormalizeFeatures = NormalizeOption.Auto;
pipeline.Add(sdcar);
var model = pipeline.Train<IncomeData, IncomePrediction>();
```

Stochastic dual coordinate ascent is a relatively simple algorithm for training a linear-form regression model. Other legacy regression trainers include FastForestRegressor, FastTreeRegressor, GeneralizedAdditiveModelRegressor, LightGbmRegressor, OnlineGradientDescentRegressor and OrdinaryLeastSquaresRegressor. Each of these has strengths and weaknesses so there's no one best algorithm for a regression problem. Understanding the differences between each type of regressor and classifier is not trivial and requires quite a bit of poring through documentation.

Once the pipeline object has been set up, training the model is a one-statement operation. If you refer back to the output shown in **Figure 1**, you'll notice that the Train method does a lot of behind-the-scenes work for you. Because the pipeline uses automatic normalization, the trainer analyzed the Age and Income columns and decided that they should be scaled using min-max normalization. This converts all age and income values to values between 0.0 and 1.0 so that relatively large values (such as an age of 52) don't overwhelm smaller values (such as an income of 4.58). Normalization usually, but not always, improves the accuracy of the resulting model.

The Train method also uses L1 and L2 regularization, which is another standard ML technique to improve the accuracy of a model. Briefly, regularization discourages extreme weight values in the model, which in turn discourages model overfitting. To reiterate, ML.NET does all kinds of advanced processing, without you having to explicitly configure parameter values. Nice!

Saving and Evaluating the Model

After the regression model has been trained, it's saved to disk like so:

```
string modelPath = "..\\..\\Models\\IncomeModel.zip";
Task.Run(async () =>
{
  await model.WriteAsync(modelPath);
}).GetAwaiter().GetResult();
```

The code assumes the existence of a directory named Model that's two levels above the program executable. An alternative is to hard code the path. Because the WriteAsync method is asynchronous,

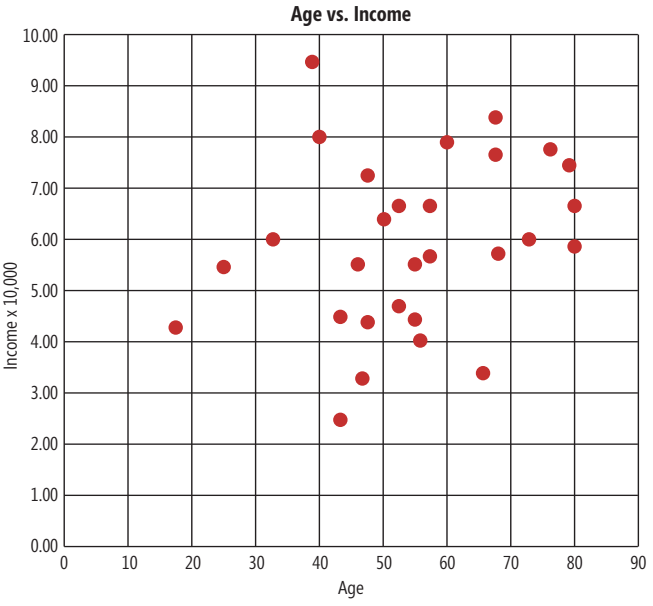


Figure 3 **Income Data**

it's not so easy to call it. There are several approaches you can use. The approach I prefer is the wrapper technique shown. The lack of a non-async method to save an ML.NET model is a bit surprising, even for a library that's in preview mode.

The model is evaluated with these statements:

```
var testData = new TextLoader(dataPath).
  CreateFrom<IncomeData>(separator: ',');
var evaluator = new RegressionEvaluator();
var metrics = evaluator.Evaluate(model, testData);
double rms = metrics.Rms;
Console.WriteLine("Model root mean squared error = " +
  rms.ToString("F4"));
```

In most ML scenarios you'd have two data files—one used just for training, and a second test dataset used only for model evaluation. For simplicity, the demo program reuses the single 30-item data file for model evaluation.

The Evaluate method returns an aggregate object that holds the root mean squared value for the trained model applied to the test data. Other metrics returned by a regression evaluator include R-squared (the coefficient of determination) and L1 (sum of absolute errors).

For many ML problems, the most useful metric is prediction accuracy. There's no inherent definition of accuracy for a regression problem because you must define what it means for a prediction to be correct. The usual approach is to write a custom function where a predicted value is counted as correct if it's within a given percentage of the true value in the training data. For example, if you set the delta percentage to 0.10 and if a true income value is 6.00, a correct prediction is one between 5.40 and 6.60.

Using the Trained Model

The demo program predicts the annual income for a 40-year-old conservative male like so:

```
Console.WriteLine("Income for age 40 conservative male: ");
IncomeData newPatient = new IncomeData() { Age = 40.0f,
  Sex = -1f, Politic = "conservative" };
IncomePrediction prediction = model.Predict(newPatient);
float predIncome = prediction.Income * 10000;
Console.WriteLine("Predicted income = $" +
  predIncome.ToString("F2"));
```

> For many ML problems, the most useful metric is prediction accuracy.

Notice that the numeric literals for age and sex use the "f" modifier because the model is expecting type float values. In this example, the trained model was available because the program just finished training. If you wanted to make a prediction from a different program, you'd load the trained model using the ReadAsync method along the lines of:

```
PredictionModel<IncomeData, IncomePrediction> model = null;
Task.Run(async () =>
{
  model = await PredictionModel.ReadAsync<IncomeData,
    IncomePrediction>(modelPath);
}).GetAwaiter().GetResult();
```

After loading the model into memory, you'd use it by calling the Predict method as shown earlier.

Figure 4 **ML.NET Legacy Example Program**

```
using System;
using Microsoft.ML.Runtime.Api;
using Microsoft.ML.Legacy;
using Microsoft.ML.Legacy.Data;
using Microsoft.ML.Legacy.Transforms;
using Microsoft.ML.Legacy.Trainers;
using Microsoft.ML.Legacy.Models;
using System.Threading.Tasks;
// Microsoft.ML 0.7.0  Framework 4.7 Build x64

namespace IncomePredict
{
  class IncomeProgram
  {
    public class IncomeData {
      [Column("0")] public float Age;
      [Column("1")] public float Sex;
      [Column("2")] public float Income;
      [Column("3")] public string Politic;
    }

    public class IncomePrediction {
      [ColumnName("Score")]
      public float Income;
    }

    static void Main(string[] args)
    {
      Console.WriteLine("Begin ML.NET demo run");
      Console.WriteLine("Income from age, sex, politics");
      var pipeline = new LearningPipeline();

      string dataPath = "..\\..\\Data\\PeopleData.txt";
      pipeline.Add(new TextLoader(dataPath).
        CreateFrom<IncomeData>(separator: ','));

      pipeline.Add(new ColumnCopier(("Income", "Label")));
      pipeline.Add(new CategoricalOneHotVectorizer("Politic"));
      pipeline.Add(new ColumnConcatenator("Features", "Age",
        "Sex", "Politic"));
      var sdcar = new StochasticDualCoordinateAscentRegressor();
      sdcar.MaxIterations = 1000;
      sdcar.NormalizeFeatures = NormalizeOption.Auto;
      pipeline.Add(sdcar);
      // pipeline.N

      Console.WriteLine("\nStarting training \n");
      var model = pipeline.Train<IncomeData, IncomePrediction>();
      Console.WriteLine("\nTraining complete \n");

      string modelPath = "..\\..\\Models\\IncomeModel.zip";
      Task.Run(async () =>
      {
        await model.WriteAsync(modelPath);
      }).GetAwaiter().GetResult();

      var testData = new TextLoader(dataPath).
        CreateFrom<IncomeData>(separator: ',');
      var evaluator = new RegressionEvaluator();
      var metrics = evaluator.Evaluate(model, testData);
      double rms = metrics.Rms;
      Console.WriteLine("Root mean squared error = " +
        rms.ToString("F4"));

      Console.WriteLine("Income age 40 conservative male: ");
      IncomeData newPatient = new IncomeData() { Age = 40.0f,
        Sex = -1f, Politic = "conservative" };
      IncomePrediction prediction = model.Predict(newPatient);
      float predIncome = prediction.Income * 10000;
      Console.WriteLine("Predicted income = $" +
        predIncome.ToString("F2"));

      Console.WriteLine("\nEnd ML.NET demo");
      Console.ReadLine();
    } // Main
  } // Program
} // ns
```

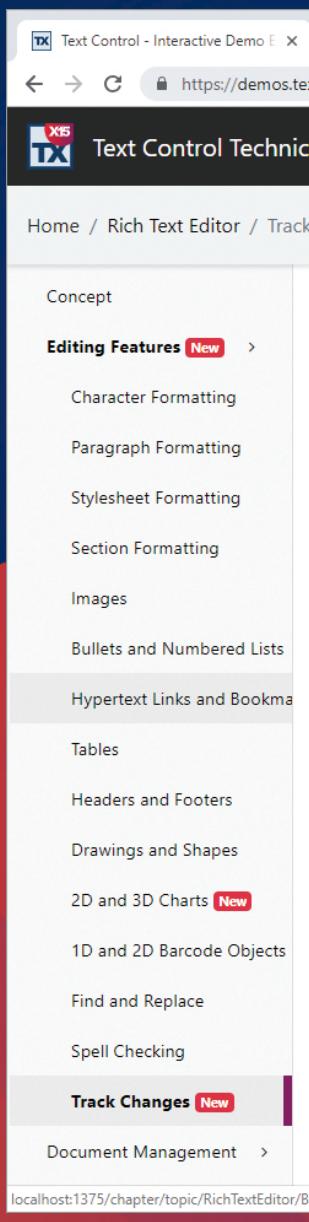
TEXT CONTROL

INTEGRATE DOCUMENT COLLABORATION

Integrate MS Word compatible track changes into cross-platform web applications. Share and review documents with a true WYSIWYG document editor.

See our technology live:

demos.textcontrol.com

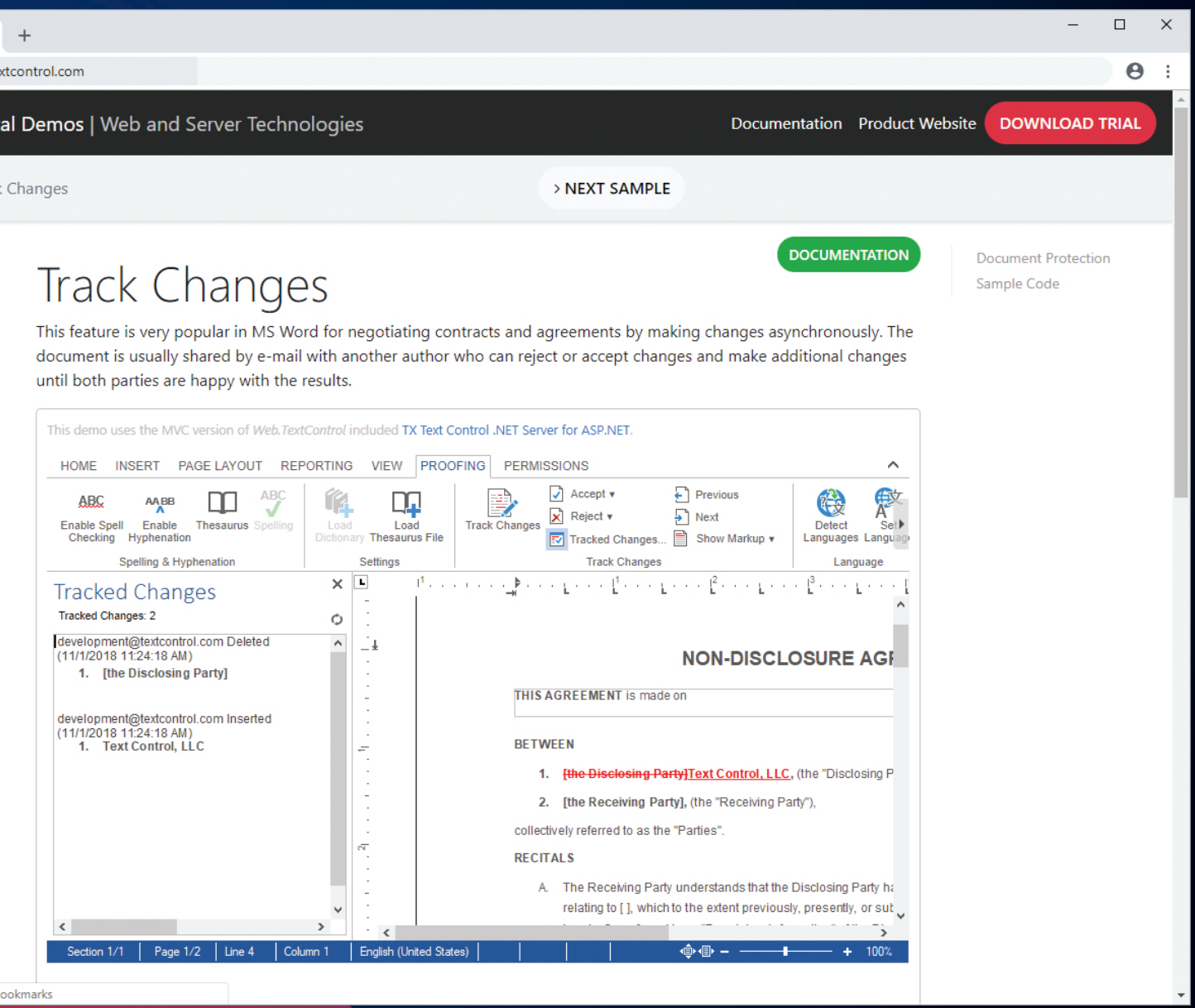


WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio Partner

TX Text Control X16 Released

Evaluate our technology and test the most sophisticated cross-browser, and true WYSIWYG, rich text editor. Merge MS Word compatible templates with JSON data and create pixel-perfect Adobe PDF documents on-the-fly. See what's possible today!



The New ML.NET API Approach

The legacy pipeline approach is simple and effective, and it provides a consistent interface for using the ML.NET classifiers and regressors. But the legacy approach has some architectural characteristics that limit the library's extensibility so the ML.NET team has created a new, more flexible approach, which is best explained by example.

Suppose you have the exact same dataset—age, sex, income, politics. And suppose you want to predict political leaning from the other three variables. A demo program to create a classifier using the new ML.NET API approach is presented in **Figure 5**. This program uses a hybrid combination of old style and new style and is intended to create a bridge between the two. Let me emphasize that the latest code examples in the ML.NET documentation will provide you with more sophisticated, and in some cases better, techniques.

Figure 5 **Classification Example Code Listing**

```
using System;
using Microsoft.ML;
using Microsoft.ML.Runtime.Api;
using Microsoft.ML.Runtime.Data;
using Microsoft.ML.Transforms.Conversions;
// Microsoft.ML 0.7.0  Framework 4.7 Build x64

namespace PoliticPredict
{
  class PoliticProgram
  {
    public class PoliticData {
      [Column("0")] public float Age;
      [Column("1")] public float Sex;
      [Column("2")] public float Income;
      [Column("3")]
      [ColumnName("Label")]
      public string Politic;
    }

    public class PoliticPrediction  {
      [ColumnName("PredictedLabel")]
      public string PredictedPolitic;
    }

    static void Main(string[] args)
    {
      var ctx = new MLContext(seed: 1);
      string dataPath = "..\\..\\Data\\PeopleData.txt";

      TextLoader textLoader =
        ctx.Data.TextReader(new TextLoader.Arguments()
      {
        Separator = ",", HasHeader = false,
        Column = new[] {
          new TextLoader.Column("Age", DataKind.R4, 0),
          new TextLoader.Column("Sex", DataKind.R4, 1),
          new TextLoader.Column("Income", DataKind.R4, 2),
          new TextLoader.Column("Label", DataKind.Text, 3)
        }
      });

      var data = textLoader.Read(dataPath);

      var est = ctx.Transforms.Categorical.MapValueToKey("Label")
       .Append(ctx.Transforms.Concatenate("Features", "Age",
         "Sex", "Income"))
       .Append(ctx.MulticlassClassification.Trainers
         .StochasticDualCoordinateAscent("Label", "Features",
         maxIterations: 1000))
       .Append(new KeyToValueEstimator(ctx, "PredictedLabel"));

      var model = est.Fit(data);

      var prediction = model.MakePredictionFunction<PoliticData,
        PoliticPrediction>(ctx).Predict(
          new PoliticData() {
            Age = 40.0f, Sex = -1.0f, Income = 8.55f
          });
      Console.WriteLine("Predicted party is: " +
        prediction.PredictedPolitic);
      Console.ReadLine();
    } // Main
  } // Program
} // ns
```

At a very high level, many ML tasks have five phases: load and transform training data into memory, create a model, train the model, evaluate and save the model, use the model. Both the legacy and new ML.NET APIs can perform these operations, but the new approach is clearly superior (in my opinion, anyway) for realistic ML scenarios in a production system.

A key feature of the new ML.NET API is the MLContext class. Notice that the ctx object is used when reading the training data, when creating the prediction model and when making a prediction.

Although it's not apparent from the code, another advantage of the new API over the legacy approach is that you can read training data from multiple files. I don't encounter this scenario often, but when I do, the ability to read multiple files is a huge time-saver.

Another feature of the new API is the ability to create prediction models in two different ways, called static and dynamic. The static approach gives you full Visual Studio IntelliSense capabilities during development. The dynamic approach can be used when the structure of your data must be determined at run time.

Wrapping Up

If you've gone through this article and run and understood the relatively simple demo code, your next step should be to take a full plunge into the new ML.NET API. Unlike many open source projects that have weak or skimpy documentation, the ML.NET documentation is excellent. I can recommend the examples at bit.ly/2AVM1oL as a great place to begin.

Even though the ML.NET library is new, its origins go back many years. Shortly after the introduction of the Microsoft .NET Framework in 2002, Microsoft Research began a project called TMSN ("text mining search and navigation") to enable software developers to include ML code in Microsoft products and technologies. The project was very successful, and over the years grew in size and usage internally at Microsoft. Somewhere around 2011 the library was renamed to TLC ("the learning code"). TLC is widely used within Microsoft and is currently in version 3.10. The ML.NET library is a descendant of TLC, with Microsoft-specific features removed. I've used both libraries and, in many ways, the ML.NET child has surpassed its parent.

This article has just scratched the surface of the ML.NET library. An interesting and powerful new capability of ML.NET is the ability to consume and use deep neural network models created by other systems, such as PyTorch and CNTK. The key to this interoperability is the Open Neural Network Exchange (ONNX) standard. But that's a topic for a future article. ■

Dr. James McCaffrey *works for Microsoft Research in Redmond, Wash. He has worked on several key 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: *Ankit Asthana, Chris Lauren, Cesar De la Torre Llorente, Beth Massi, Shahab Moradi, Gal Oshri, Shauheen Zahirazami*

Manipulating Documents?

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

Try GroupDocs APIs for FREE

Download a Free Trial at

https://downloads.groupdocs.com

GROUPDOCS

GroupDocs.Viewer

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

GroupDocs.Annotation

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

GroupDocs.Conversion

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

GroupDocs.Comparison

Compare two documents and get a difference summary report.

GroupDocs.Signature

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

GroupDocs.Assembly

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

GroupDocs.Metadata

Organize documents with metadata within any cross platform application.

GroupDocs.Search

Transform your document search process for advance full text search capability.

- GroupDocs.Text
- GroupDocs.Editor
- GroupDocs.Parser
- GroupDocs.Watermark

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

Accelerate AI Solutions with Automated Machine Learning

Krishna Anumalasetty

Machine learning (ML) is being used in a wide range of applications, from autonomous cars and credit card fraud detection to predictive maintenance in manufacturing and beyond.

But there's a problem. Building ML solutions is complex and requires highly skilled personnel with Ph.D.s in mathematics or other quantitative fields. The demand for data scientists has outpaced supply, inhibiting adoption of ML among enterprises. Many companies have vast stores of data, yet they're unable to employ predictive analytics to improve business decision making and achieve success.

The automated ML capability in Azure Machine Learning is designed to overcome these obstacles and make AI more accessible to every developer and every organization. In this article, I'll show how automated ML can be used to quickly build an energy demand forecasting solution.

This article discusses:

- The stages of the data science lifecycle process and how it maps to automated ML
- Building an energy demand forecasting app using data from publicly available sources
- Accelerating AI by automating feature engineering, algorithm selection and hyperparameter tuning
- Configuring automated ML and training models to generate a leaderboard of the most effective models

Technologies discussed:

Automated Machine Learning, Azure Machine Learning Service, Azure Cloud Compute

The Machine Learning Lifecycle

Look at the data science lifecycle process shown in **Figure 1**. It breaks the lifecycle into four stages: Business Understanding, Data Acquisition & Understanding, Modeling, Deployment. Every ML solution should start with the business problem you're working to solve—that is, the Business Understanding stage. Next, you acquire and explore the data—the Data Acquisition & Understanding step—to bring in the raw data that may be in various data sources in different formats. Ingesting, exploring, and de-duping are some of the activities in Data Acquisition & Understanding.

The Modeling stage has three steps, starting with the feature-engineering process. Here you may transform the data into new features and generate entirely new features. Features are the predictors that influence prediction. An easy way to understand features is to take the example of estimating a house price, which is influenced by the size and location of the house, the number of bedrooms, and other factors. So, if you're generating an ML model to predict a house value, these factors are the features.

Next comes the model training process (in the Modeling stage) where you build ML models by applying many different algorithms and hyperparameters to the dataset. This process involves evaluating the different models you've built to choose the one that works best for the current application. From there, you test and finally deploy the model into production, where it's used to run predictions from the new data.

Once in the Deployment stage, you must monitor the model for drifts, retraining it periodically when drifts reach a certain threshold or when new data makes clear that the model isn't performing to expectation.

Now, let's see how automated ML helps build an ML model. The first step is to install the set of Azure Machine Learning Python

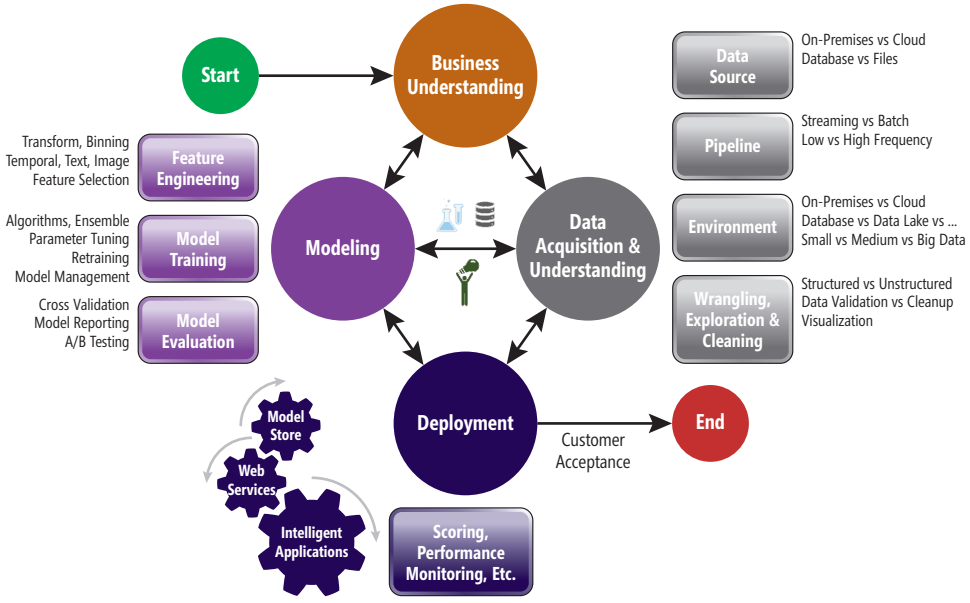


Figure 1 **Data Science Lifecycle Process**

libraries. Automated ML libraries are included and part of Azure ML libraries and will be used in your ML scripts.

As a developer, if you're familiar with languages such as C# or Java, you should be able to get up to speed in Python in no time. You can install the Azure ML SDK, which includes automated ML libraries on your computer, however, for simplicity we'll use Azure Notebooks, which is an Azure service to host Jupyter Notebooks. Jupyter Notebook is an interactive notebook environment popular with data scientists for ML solution development. You can learn more about using Jupyter Notebooks from Frank La Vigne's Artificially Intelligent column in the February 2018 issue of *MSDN Magazine* (msdn.com/magazine/mt829269).

Automated ML libraries are free to use and come pre-installed on the Azure Notebooks service, eliminating the need for a setup process. Sign in to Azure Notebooks using your Microsoft Account. If you already have an Azure Subscription, you can use the Azure Active Directory (Azure AD) account (which may be the same as your corporate credentials if your organization federated with Azure AD).

Use your Azure subscription ID to setup an ML workspace that you can share and collaborate with your teammates either through the Azure Portal, or through the API, like so:

```
ws = Workspace.create(name = "energy_ml_ws",
                      subscription_id = "<azure_subscription_id>",
                      resource_group = "energy_ml_rg",
                      location = "eastus",
                      create_resource_group = True,
                      exist_ok = True)
```

Energy Demand Forecasting

With workspace creation complete, let's work on the business problem—an energy demand forecasting example that features time series data. Time series forecasting is the task of predicting future values in a time-ordered sequence of observations. It's a common problem and has applications in many industries. For example, retail companies need to forecast future product sales so they can organize their supply chains to meet demand. Similarly, package delivery companies need to estimate the demand for their services so they can plan workforce requirements and delivery routes ahead of time. In many cases, the financial impact due to inaccurate forecasts can be significant, making forecasting a business-critical activity.

This is certainly true for energy utilities, which must maintain a fine balance between the energy consumed on the grid and the energy supplied to it. Purchase too much power and the operator must store the excess energy, which is expensive. Purchase too little and it can lead to blackouts, leaving customers in the dark. Grid operators can make short-term decisions to manage energy supply to the grid and keep the load in balance, but an accurate forecast of energy demand is essential for operators to make these decisions with confidence.

Data Acquisition

Let's walk through how you can use ML to build an energy demand forecasting solution. There's a public dataset available from the New York Independent System Operator (NYISO), which operates the power grid for New York State. The dataset has hourly power demand data for New York City over a period of five years.

The NYISO dataset has a timestamp and energy demand in MWh at hourly increments from 2012 to 2017. To inspect and explore the data, let's first read the data that's in Azure Blob Storage into a Pandas dataframe. Pandas dataframes are popular with the ML community and offer an easy way to manipulate data. Here's the code for the import:

```
import pandas as pd
demand = pd.read_csv(
  "https://antaignitedata.blob.core.windows.net/antaignitedata/nyc_demand.csv",
  parse_dates=['timeStamp'])
```

Let's print the first few rows of the data by invoking the head method on the dataframe, which prints the top few rows and produces a simple table with timestamps and energy demand values. If I plot the chart of energy demand over a week in July 2017, it results in a line chart that shows fluctuating hourly values over the course of seven days. For this I use matplotlib, a library that helps easily plot charts within Jupyter Notebook. Here's the code:

```
plt_df = demand.loc[(demand.timeStamp>'2016-07-01') & (demand.
timeStamp<='2016-07-07')]
plt.plot(plt_df['timeStamp'], plt_df['demand'])
plt.title('New York City power demand over one week in July 2017')
plt.xticks(rotation=45)
plt.show()
```

Of course, weather has a direct impact on energy consumption. On a hot day, use of air conditioning will significantly increase demand for electricity. So an additional dataset containing hourly weather conditions in New York City over the same time period can be used

to improve prediction. I can access weather data from darksky.net. Now, I'll read the weather dataset into a dataframe and augment the NYISO dataset with weather data by merging the two datasets, using this code:

```
weather = pd.read_csv(
  "https://antaignitedata.blob.core.windows.net/antaignitedata/nyc_weather.csv",
  parse_dates=['timeStamp'])
df = pd.merge(demand, weather, on=['timeStamp'], how='outer')
```

The resulting merged dataset will look something similar to the table in **Figure 2**, effectively adding weather information—precipitation and temperatures—to the NYISO dataset. To see the first few rows of the data, invoke the head method on the dataframe.

Training dataset is used to train the model.

Modeling

Now we come to the modeling step, which involves three disciplines: feature engineering, model training and model evaluation. This is where data scientists spend most of their time, and it's where automated ML can really help simplify things.

The dataset I'm using is time series data, which requires me to generate features. Typical features for time series data are date time features, lag features and window features. To keep things simple, I won't go into lag features or window features. Automated ML automatically generates date time features, eliminating the need to manually generate these. It also performs many data preprocessing tasks. For example, automated ML can impute missing values in the dataset. If I inspect the dataset further I'll find that there are a few missing rows in the dataset and a few missing values, as well. In the following code, you can see that row 49175 contains a missing value, indicated by "nan," which stands for "not a number":

```
df.iloc[49175].values
```

Output of the line is as follows:

```
array([Timestamp('2017-08-11 01:00:00'), nan, 0.0, 69.26], dtype=object)
```

Next, I need to split the dataset into a training dataset and a test dataset. The training dataset is used to build the model. Once trained, I need to test the model using the test dataset to evaluate the performance and to ensure the model is satisfactory before deployment. The splitting can be random, but in the case of energy consumption data where there's seasonality, I would want to split the dataset strategically. A small portion of training dataset is set aside as a validation dataset. Training dataset is used to train the model. Validation dataset is used to generalize the model so that the model isn't overfitted with the training data. This helps to make sure the model also performs well on new data and not just on the training data.

Now, the column that I'm working to predict is demand. I move the demand values into its own vector "y," which is called the label column. I'm setting aside as my test dataset any data that's newer than July 1, 2017. All data that's older than July 1, 2017, is the training data. The code here shows this:

```
train, test = (df.loc[df['timeStamp']<'2016-07-01'],
  df.loc[df['timeStamp']>='2016-07-01'])
```

The data that was read into the dataframe has the features and label (the column I'm trying to predict). I need to move the label column into its own vector y, like so:

```
X = train.drop(['demand'], axis=1)
y = train['demand']
y = y.values
```

Set aside a validation dataset within the train dataset. With timeseries data, the validation dataset is typically for a specific period. Here's how the split is done:

```
split_index = int(X.shape[0] * 0.9)
X_train = X[0:split_index]
y_train = y[0:split_index]
X_validation = X[split_index+1:]
y_validation = y[split_index+1:]
```

The feature engineering part of the Modeling has been completed.

Model Training

The next choice a user faces is deciding which algorithm works for the dataset. Complexity can be a problem here as there are many algorithms to choose from, including support vector machine (SVM), lasso regression, ridge regression and more. What's more, each algorithm has hyperparameters that must be tuned. Hyperparameters for each of the algorithms can be an infinite set, which means the combination of algorithms and hyperparameters is itself infinite. So, theoretically you would need to build an infinite number of models to find the best one.

Users combine features, learners (algorithms) and hyperparameters to build multiple models for a given business problem, so they can ultimately find one that yields optimal accuracy. Building many models and evaluating them is a manual, time-consuming and tedious task that can take weeks or even months. On top of that you need to maintain the solutions and the models that are deployed. As data evolves, you need to periodically perform the model building process again. Building a single model involves the tasks of feature engineering, algorithm selection and hyperparameter tuning. To get a good performing model, you need to build and evaluate several models and perform these tedious tasks repeatedly.

For those that are experienced in data science, automated ML simplifies and eliminates manual processing by performing feature engineering, algorithm and hyperparameter selection, and tuning for you to improve productivity and save time. For new data scientists, the abstraction of algorithm selection and hyperparameter tuning simplifies the complexity and helps you build ML solutions quickly.

As a user I would like to save all the ML models I've created, as well as the history of all the training jobs, so I have a record of how I got here and can refer back to it when needed. To do this, let's create a project folder where all the artifacts get stored, and an experiment object to associate all the run history of automated ML training jobs. First, I create a project folder to store all the project-related files, like so:

```
project_folder =
  './sample_projects/automl-energydemandforecasting'
```

	timeStamp	demand	precip	temp
0	2012-01-01 00:00:00	4937.5	0.0	46.13
1	2012-01-01 01:00:00	4752.1	0.0	45.89
2	2012-01-01 02:00:00	4542.6	0.0	45.04
3	2012-01-01 03:00:00	4357.7	0.0	45.03
4	2012-01-01 04:00:00	4275.5	0.0	42.61

Figure 2 **Energy Demand Dataset Merged with Weather Dataset**

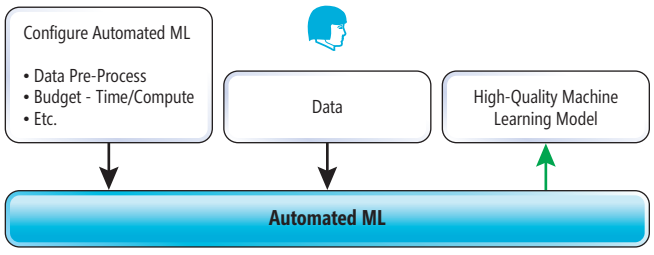


Figure 3 **Conceptual Automated Machine Learning Diagram**

Then, I'll choose a name for the run history container in the workspace, with this code:

```
experiment_name = 'energy_ml_exp'
```

Next, I'll create an experiment object and associate it with the workspace. Here's the code:

```
experiment=Experiment(ws, experiment_name)
```

From there, automated ML needs only two steps to build the models before I configure the AutoMLConfig object and run the experiment. **Figure 3** shows the conceptual representation of automated ML.

Now, let's configure the automated ML settings and submit an automated ML experiment. Here's the code:

```
automl_config_local = AutoMLConfig(task = 'regression',
                                   debug_log = 'automl_errors.log',
                                   primary_metric = 'spearman_correlation',
                                   iterations = 150,
                                   X = X_train,
                                   y = y_train,
                                   X_valid = X_validation,
                                   y_valid = y_validation,
                                   preprocess = True,
                                   path=project_folder)
```

Notice in the config settings that I set iterations = 150. Because I set the number of iterations to 150, automated ML will generate 150 different combinations of algorithms and hyperparameters, which culminate into 150 different models. Automated ML itself uses an ML model that was trained with millions of pipelines. Using Matrix factorization techniques, automated ML generates the algorithms and hyperparameter combinations in an intelligent way, leading to much faster convergence on an optimal model. Also, setting preprocess = True triggers automated ML to perform a data preprocessing and feature engineering tasks. This includes generating the date time features, imputing missing values, converting categorical values into one hot encoding, and the like—all of which eliminate the need for a data scientist to do these manually.

Now you can complete the second step, which is submitting the automated ML experiment, like so:

```
local_run = experiment.submit(automl_config_local, show_output=True)
```

At this point automated ML generates a set of algorithm and hyperparameter combinations. Training of the models is done on Azure Virtual Machine managed by Azure Notebooks service. Automated ML gives you the choice of running the model training jobs on your local computer, or in Azure Cloud to scale up and scale out as needed for additional performance. You can run these iterations in parallel on a cluster. You can monitor the progress of the runs in Azure Portal, or in the Jupyter Notebook through the widget extension that comes with the SDK. Automated ML evaluates the generated models based on your criteria, and can render a leaderboard that shows the models in order of performance.

You can also review a variety of charts for each of the models and inspect different metrics to help make decisions. In cases where accuracy is extremely important, you can further tune the generated model manually to improve its performance.

All the models generated by automated ML are stored in durable storage in Azure, as a serialized python object in a format called PKL.

Model Evaluation and Testing

Now that automated ML has generated a high-quality model, let's use it to run predictions on the test data that was set aside. First, I need to select the best model from all the models that were generated, using the following code:

```
best_run, fitted_model = local_run.get_output()
print(best_run)
print(fitted_model)
```

That yields the following code, describing the best model generated by automated ML:

```
Run(Experiment: enery_ml_exp, Id: AutoML_dc619226-fbdb-41e6-872c-6c59ab2b5209_1,
  Type: None, Status: Completed) Pipeline(memory=None, steps=[('datatransformer',
  DataTransformer(logger=None, task=None)), ('sparsenormalizer',
  <automl.client.core.common.model_wrappers.SparseNormalizer object at
  0x7f7f87ee3828>), ('decisiontreeregressor',
  DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=0.2,
  max_leaf_node... min_weight_fraction_leaf=0.0, presort=False,
  random_state=None, splitter='best'))])
```

Notice that though I've been calling these models, they are in fact pipelines that include all the preprocessing transformation through which the data must go. Now let's test the model by running predictions on the test data to produce a charted comparison of actual data against predicted data. Use the following code:

```
x_test = test.drop(['demand'], axis=1)
y_test = test['demand']
y_pred_test = fitted_model.predict(x_test)
y_residual_test = y_test - y_pred_test

plt.plot(x_test['timeStamp'], y_test, label='Actual')
plt.plot(x_test['timeStamp'], y_pred_test, label="Predicted")
plt.xticks(rotation=90)
plt.title('Actual demand vs predicted for test data ')
plt.legend()
plt.show()
```

With a little more feature engineering and the use of lag features, this model can be improved even further. Moving forward, we can expect automated ML to evolve to automate more and more areas of feature engineering, including lag feature generation.

Wrapping Up

Automated ML lets you build an ML model with just a few simple steps. We're in the early stages of making ML model building accessible to developers to truly democratize AI. You can find many more samples and tutorials of using automated ML at aka.ms/AutomatedMLDocs. ■

Krishna Anumalasetty *has been working as a program manager in Azure and cloud services for the last seven years, with four of those in ML and AI. Anumalasetty is the program manager that incubated automated ML as part of the AI stack in Azure. He has worked on enabling enterprise excellence capabilities, such as encryption @REST data protection, VNET capabilities, hybrid scenarios with on-premises connectivity, Azure Active Directory integration and more. He is currently working on enriching the automated ML capabilities with deep learning.*

Thanks to the following Microsoft technical experts for reviewing this article: *Sujatha Sagiraju, Bharat Sandhu*

Deploy Your Code the Right Way with Azure Pipelines

Micheal Learned and Andy Lewis

Modern applications are increasingly complex systems that involve multiple technology stacks and cloud-native services. Orchestrating an automated release pipeline for these systems can be challenging. Azure Pipelines provides powerful, easy-to-use continuous integration (CI) and continuous delivery (CD) services you can use to build and test your app and then deploy to your intended targets. In this article, we'll provide an overview of the key concepts of Azure Pipelines and discuss deployment scenarios for various Azure services. We'll also walk through a detailed scenario for creating a pipeline for a .NET Core app that targets Docker containers in Azure Kubernetes Service (AKS). Finally, we'll show you the next generation of CI/CD in which your YAML pipeline is configured as part of your code.

Azure Pipelines provides flexibility for configuring your build and release workflows. You can choose from multiple version control providers such as GitHub, Azure Repos, Bitbucket and Subversion. Most languages and application frameworks are supported, including .NET, Java, JavaScript, Python, PHP, Ruby, Xcode, C++ and Go. You can create pipelines that deploy anywhere, including any cloud provider or on-premises deployment target. You can use any package management repository such as NuGet, npm, and Maven to produce and consume packages. Azure Pipelines allows you to configure and customize your CI/CD solution to fit virtually any scenario. Built-in pipeline tasks, third-party tasks, custom tasks, and file-based or inline PowerShell and Bash scripting all help you build workflows for your pipelines. Templates provide you with an easy way to quickly group and reuse a set of tasks.

This article discusses:

- Continuous integration and continuous delivery
- Configuration as code
- Azure deployment

Technologies discussed:

Azure Pipelines, Azure Kubernetes Service, YAML

Azure Pipelines Key Concepts

In Azure Pipelines you define build (CI) and release (CD) workflows for your apps. These workflows provide a framework for orchestrating and automating your release pipelines. A typical scenario is for your CI build pipeline to pull code from your version control repository, compile the code, run tests and other analysis processes, and finally publish artifacts, such as files, binaries, and executables. A CD release pipeline is automatically triggered and uses the published artifacts as the deployment payload for testing and production environments. Azure Pipelines allows you to create simple or complex CI/CD pipelines to deploy your apps to a wide range of deployment targets, including a variety of Azure services such as Web servers and databases. You define various stages of a release pipeline such as Dev, QA and Production. It's important to note that build and release pipelines are flexible, and you can define pretty much any workflow you want with them.

Agents are Windows, Linux, and macOS machines (or containers) that run the jobs in your build and release pipelines. You can install and self-host your agents on physical or virtual machines (VMs) either on-premises or in the cloud (for example, Azure). Or to save yourself time, you can use Microsoft-hosted Linux, macOS and Windows agents. For open source projects, we provide up to 10 parallel Microsoft-hosted jobs for free (see the links at the end of this article for details).

Build and release tasks are the basic building blocks of your pipelines. They're packaged scripts that do the work in your pipeline. Azure Pipelines comes with many common tasks for performing actions to build, test, package and deploy code. You can deeply customize your process with tasks that support file-based or inline Bash, PowerShell, and Python scripts. You can even run your Windows batch file scripts. You can use tasks provided by Azure Pipelines, leverage third-party tasks from Visual Studio Marketplace and, if necessary, create your own custom tasks.

Build and release templates provide reusable patterns to help you quickly configure CI/CD for common scenarios such as building an Android app or deploying an Azure Web app. A template is essentially a pre-defined collection of tasks with parameters preset to meet the needs of common app types and targets. You can customize the templates and even put your custom workflows into reusable templates to support the specific needs of your organization.

Service connections provide a secure mechanism for connecting external services to Azure Pipeline tasks. These external services can include your Azure subscriptions, Jenkins or other CI systems, and remote services such as file servers, to name just a few. A typical scenario is to create an Azure Resource Manager (ARM) service connection to enable Azure Pipelines to deploy your apps to various Azure service targets, such as Web Apps, Azure Virtual Machines, Azure SQL Database or any other resource in your Azure subscriptions.

Azure Deployment Targets

Azure provides a variety of services that serve as deployment targets for your apps. App Service, Windows or Linux virtual machines, containers, and many other Azure services are valid deployment targets. Docker containers can be used with App Service, VMs and with Kubernetes via AKS.

Virtual machines (Linux and Windows) are common deployment targets for a wide range of apps. Azure Pipelines supports VMs as deployment targets in Azure, on-premises or even on other cloud providers. You install and configure a deployment agent on your VMs, and then you can hook up Azure Pipelines deployments to your VMs. Agents and Azure Pipelines communicate over asymmetric encryption and HTTPS to provide a secure deployment approach. Deployment groups allow you to organize the servers that host your app. Each node in the deployment group hosts an Azure Pipelines agent, and you can deploy to them. One example for the use of deployment agents is to deploy Web app code to a group of Web servers. You can also install agents on physical machines and use Azure Pipelines to deploy to them with similar patterns.

App Service is an Azure service that allows you to host applications in the language of your choice without having to manage infrastructure, and you can run APIs, mobile back ends, Web Apps and serverless code. Web App for containers allows you to run containerized applications on Windows and Linux. There are many ways to deploy apps to App Service, including with Azure Pipelines. You can use existing Azure Pipelines templates to quickly build a CD pipeline for your app. The App Service Deploy task provides a variety of standard features such as deploying to specific App Service slots, overriding config values at deploy time and executing customs scripts.

Azure services and infrastructure can be deployed with templates to provide infrastructure as code (IaC), which yields many benefits such as leveraging version control, CI/CD and automating the deployment of the infrastructure for your apps.

AKS provides an Azure-based Kubernetes management service that allows you to deploy and manage containerized applications easily. AKS removes the administration overhead for managing complex container orchestration and simplifies container management by handling health monitoring and maintenance for you. Azure manages Kubernetes masters, and you manage and pay only for the agents. AKS integrates with Azure Container Registry to provide a fully managed container solution.

Azure services and infrastructure can be deployed with templates to provide infrastructure as code (IaC), which yields many benefits such as leveraging version control, CI/CD and automating the deployment of the infrastructure for your apps. ARM templates provide a declarative JSON approach for deploying Azure infrastructure. You can define your networks, security groups, load balancers, AKS and many other Azure infrastructure components with ARM templates.

Azure Pipelines provides support for deploying ARM templates with the Azure Resource Group Deployment task. The task allows you to create pipelines that leverage ARM templates to deploy Azure services to your subscriptions. The task provides several useful configuration options such as the ability to validate templates to prevent errors, deploy in incremental mode to avoid impacting existing Azure resources, and handle creating or updating existing resource groups.

SQL Server is a database deployment target that comes in two forms in Azure. SQL Server can be self-hosted, and Azure SQL Database is an Azure service for Database as a Service (DBaaS) that allows you to use SQL Server without managing infrastructure.

Azure Pipelines can deploy to both forms of SQL Server. SQL Server provides several options for deploying schema changes with tasks such as support for dacpac, executing scripts with the sqlcmd utility and using third-party services with Redgate tooling.

Example: Deploy a .NET Core App to an AKS Container Target

.NET Core is a cross-platform (Windows, Linux and macOS) development platform for building device, cloud and Internet of Things (IoT) applications. .NET core is maintained and supported by Microsoft, RedHat and the .NET community. AKS is a managed Kubernetes service for Azure. Kubernetes simplifies the use of containers. You can create a CI/CD pipeline with Azure Pipelines targeting AKS with Docker containers. ARM templates are used to represent the Azure infrastructure, such as AKS and a container registry. Docker images are used to containerize an ASP.NET Web App. This is a moderately complex pipeline, so to help you get started quickly, you'll find a detailed walk-through that automates the creation of the CI/CD pipeline for this scenario at bit.ly/2T9Y0pT.

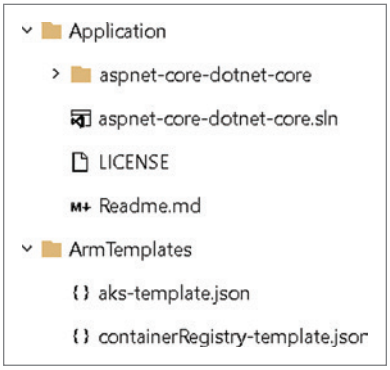


Figure 1 **ASP.NET Core App and ARM Templates**

An ASP.NET Core Web app is stored in Azure Repos in this example. The ARM templates represent the AKS cluster and the container registry. Using IaC this way allows you to deploy the Azure infrastructure alongside the application, therefore you're able to manage the Azure Resources and the application in a single CI/CD pipeline. **Figure 1** is a view of the basic structures in Azure Repos.

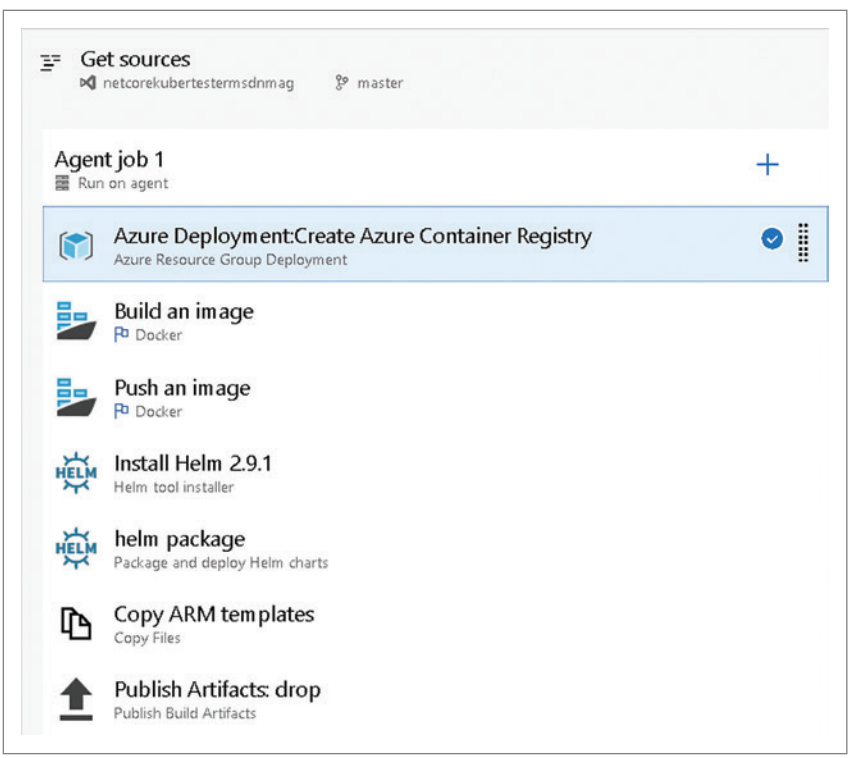


Figure 2 **Build Pipeline with Docker and AKS**

The Build pipeline for this scenario contains several steps. You can name the tasks with descriptive labels. **Figure 2** shows a set of tasks for setting up a containerized ASP.NET Core app that targets an AKS cluster.

Get Sources is the first step in the build process. The ASP.NET Core App and the ARM templates are fetched from Azure Repos and brought to the build agent. Cloud-hosted agents keep you from having to manage infrastructure for executing pipelines.

Creating an Azure Container Registry is the next step in the build process. The registry stores and manages Docker images. The next few steps build and publish the Docker image, which in this case contains an ASP.NET Core application. This build step uses an Azure Resource Group Deployment task to deploy an ARM template, which represents the container registry. This task is an example of the flexibility of Azure Pipelines as you're able to execute a deployment task to set up the registry as part of the build process. The ARM deployment task is set to "incremental mode" by default, so if the container registry already exists, it won't get created again on subsequent build runs.

Docker tasks are used to build and publish Docker images to the container registry. This step containerizes the ASP.NET Core app. Using an immutable infrastructure with a portable container for the app enables deployment of the container to a wide variety of targets on Windows, most Linux distros and macOS. In this scenario, the container is targeting AKS in Azure.

Helm Charts provide the packaging format for Kubernetes that make it easy to define, version and install Kubernetes applications. Helm, a Kubernetes package manager, is installed on the build agent with the Helm tool installer task. The package and deploy Helm Charts task archives the chart on this step so that the Helm Charts end up published as build artifacts.

Copy ARM templates, which is a Copy task, copies the templates to a staging directory on the build agent. ARM templates are JSON files that represent Azure resources. The ARM template here represents the Kubernetes cluster resource. The ARM templates are stored in Azure Repos, but you can optionally use another version control provider, such as GitHub, for your pipelines.

Publish Build Artifacts is a task that copies the contents of the staging directory to Azure Pipelines. You can store the artifacts on a file share somewhere, but Azure Pipelines are convenient and require no additional set up. The artifacts, in this case, are the ARM templates and the Kubernetes package produced by Helm Charts.

After the build pipeline executes, the release pipeline consumes the artifacts from your build

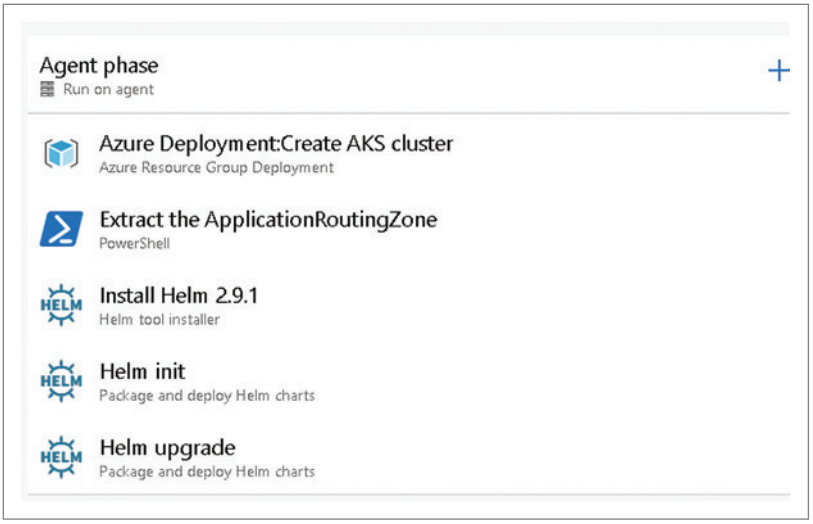


Figure 3 **Release Pipeline for .NET Core App and Kubernetes**

pipeline and deploys the solution. **Figure 3** shows the various steps in the release pipeline.

The Azure Resource Group Deployment task is the first step in the release pipeline. This task consumes the AKS ARM template and is the artifact used to deploy the AKS cluster. The ARM template artifacts are available for the release pipeline here because the templates were published as artifacts as part of the build pipeline run.

PowerShell tasks can be used in any build and release pipeline. The PowerShell task here uses the setvariable logging command to initialize some persistent variables so that key bits of common data can be accessed by the subsequent Helm tasks. In this case the script accesses deployment outputs that were created by the previous ARM deployment task. Specifically, variables are set for later use for application routing and Application Insights:

```
$deploymentOutputs=(ConvertFrom-Json '$(deploymentOutputs)')
$applicationRoutingZone=$deploymentOutputs.applicationRoutingZone.value
$aiKey=$deploymentOutputs.aiKey.value
Write-Host "##vso[task.setvariable
  variable=applicationRoutingZone;]$applicationRoutingZone"
Write-Host "##vso[task.setvariable variable=aiKey;]$aiKey"
```

Helm tasks orchestrate the final three steps of the release process. The Helm tool installer ensures the agent installs the latest version of the Kubernetes package manager, which includes prerequisites such as Kubectl, the command-line tool for Kubernetes. Helm init and Helm upgrade commands are executed to deploy the application to Kubernetes. Once the release pipeline executes, the ASP.NET Core application is deployed to a Kubernetes cluster, and the application is accessible via URL.

As an option to quickly get started, Azure DevOps Projects can help you automate the creation of a CI/CD pipeline like the one described earlier, especially if you're not familiar with deploying to Azure services. You use a wizard-driven experience from the Azure portal to choose from a variety of application frameworks and deployment targets. In a few simple steps, you can create a fully functional CI/CD pipeline that builds and deploys your code to various Azure services, such as App Service, VMs, AKS and SQL Database. DevOps Projects allow you to bring your own code or use any of several provided sample applications. You can use DevOps Projects to create a pipeline in Azure Pipelines, and then use that pipeline as a reference architecture for how to do CI/CD for specific application frameworks and Azure service deployment targets. You can also further customize the pipelines. Azure DevOps Projects also creates additional Azure resources such as Application Insights resources for monitoring and an Azure Portal-based dashboard. Azure DevOps Projects documentation can be found at bit.ly/2B7XnpJ.

Config as Code in a YAML Pipeline

In the previous example, you saw how you can use the Azure Portal to generate an end-to-end build pipeline that creates and then feeds artifacts into a release pipeline. The types of pipelines the Azure Portal generates are called "designer pipelines"; these are the drag-and-drop pipelines. This year we introduced a new kind of pipeline you can define in your source code: a YAML pipeline.

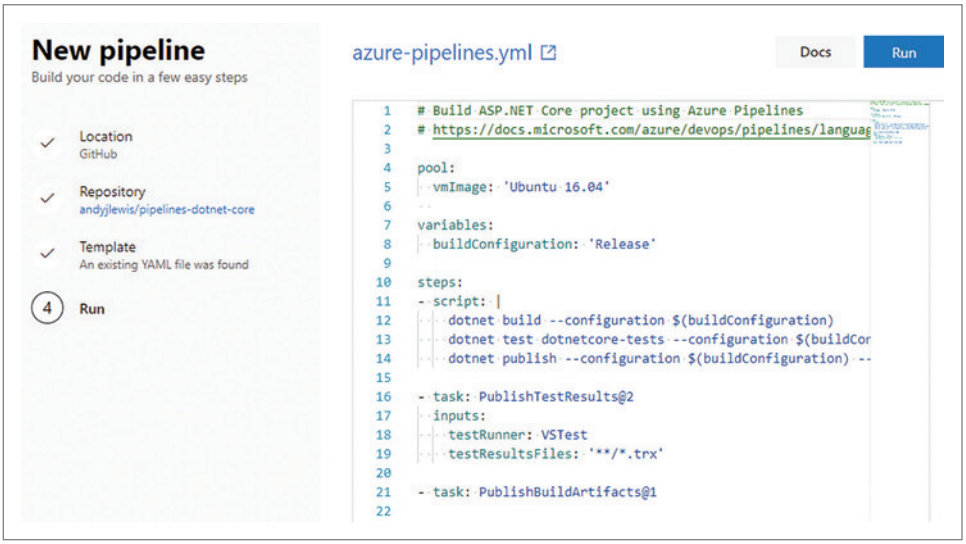


Figure 4 **A Basic ASP.NET Core YAML Pipeline in a Sample App Repo**

The most obvious advantages of a YAML pipeline come from the fact that the workflow lives in your code. You can branch, diff and merge changes to your business logic. If you think that's cool, wait until the next time someone changes your pipeline and causes a build break or an unexpected outcome. As painful as this scenario can be, you'll find relief in the fact that you can spot, track and fix the problem just like any other bug in your code!

If your repo contains an azure-pipelines.yml file at the root level, then when you go to

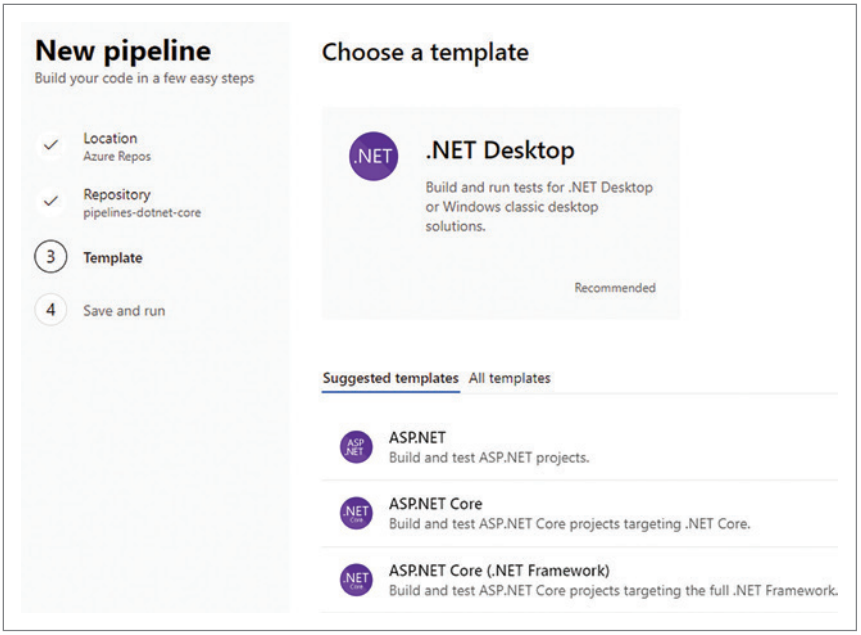


Figure 5 **Azure Pipelines Looks at Your Code and Suggests a Template for Your YAML Pipeline**

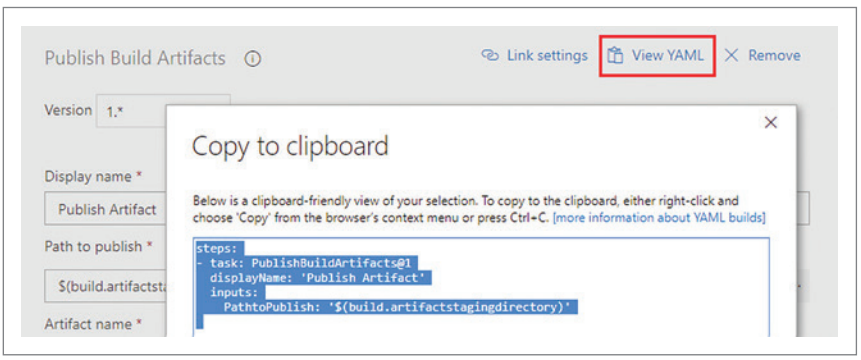


Figure 6 **After You Select a Task in the Designer You Can View the YAML**

To help you migrate from designer pipelines and learn how to adapt them to YAML, tools and information are provided. When you edit your designer pipeline, select a task, and then select View YAML, as shown in **Figure 6**.

In some cases, you can use this snippet directly. Sometimes (for example if your task uses process parameters, usually generated by a template), you'll need to manually adjust the YAML.

There's also a VS Code extension in preview at aka.ms/azure-pipelines-vscode. It's powered by a language server in case you want to build pre-commit hook tools. Find out more at aka.ms/azure-pipelines-language-server.

To learn more about what you can do with YAML, see aka.ms/azure-pipelines-yaml-schema and aka.ms/azure-pipelines-task-reference.

Scripts are a very common way to handle CI and CD workflows in code. YAML pipelines are designed to make it as easy as possible for you to integrate scripts wherever you like, in as portable a way as possible. As you get up to speed in YAML, you'll probably notice both task-centric and script-centric approaches. The decision about which approach to use really comes down to your personal preference.

For simple tasks, you can use a generic cross-platform script:

```
steps:
- script: echo This is pipeline $(System.DefinitionID)
```

You can explicitly run a PowerShell or Bash Script. Bash scripts have the advantage of being runnable on multiple platforms. See **Figure 7** for an example. To learn more, see aka.ms/cross-platform-scripts.

create the pipeline, the system picks up the file. You can fork one of our sample app repos to see this in action at aka.ms/get-started-yaml-pipeline. **Figure 4** shows a basic ASP.NET Core YAML pipeline. If your GitHub repo doesn't already have an azure-pipelines.yml file in it, then when you create a new pipeline, the system analyzes the code in your repo and then suggests the kind of pipeline you need, as shown in **Figure 5**.

Figure 7 **YAML Pipeline with a Bash Script That Can Run on Linux, macOS and Windows Agents**

```
trigger:
  batch: true
  branches:
    include:
    - master
steps:
- bash: |
   echo "Hello world from $AGENT_NAME running on $AGENT_OS"
   case $BUILD_REASON in
    "Manual") echo "$BUILD_REQUESTEDFOR manually queued the build." ;;
    "IndividualCI") echo "This is a CI build for $BUILD_REQUESTEDFOR." ;;
    "BatchedCI") echo "This is a batched CI build for $BUILD_REQUESTEDFOR." ;;
    *) $BUILD_REASON ;;
    esac
  displayName: Hello world
```

If your team doesn't need the stages and approval options provided by a designer release pipeline, then you can build, test and deploy all within a single YAML pipeline, as shown in **Figure 8**.

This example works for Linux and Windows Web apps. To adapt it for Windows, just add and remove the relevant comments to change the Microsoft-hosted pool you're using, and to set the TakeAppOfflineFlag parameter to true to take the app offline before deployment to avoid a file-in-use error on your Web app .DLL file.

Service connections (formerly "endpoints") can be a tricky part of the process to set up. The following parameters are how you specify the target for your Web app:

```
azureSubscription: 'YourSubscription'
```

and

```
WebAppName: 'your-webapp'
```

These parameters refer to fields in an Azure Resource Manager service connection. See aka.ms/azure-pipeline-service-connection.

Wrapping Up

Getting started with Azure Pipelines is free. Azure Pipelines provides various free usage tiers for a certain number of monthly minutes

Figure 8 **Build, Test and Deploy from a YAML Pipeline That Lives in Your Codebase**

```
# Build, test and deploy in a YAML pipeline

pool:
  vmImage: 'ubuntu-16.04'
  # replace the hosted Ubuntu pool above with the Windows pool below if
  # you want to deploy to a Windows web app
  # vmImage: 'vs2017-win2016'

variables:
  buildConfiguration: 'Release'

trigger:
- master

steps:
- script: |
    dotnet build --configuration $(buildConfiguration)
    dotnet test dotnetcore-tests --configuration $(buildConfiguration) --logger trx

- task: PublishTestResults@2
  inputs:
    testRunner: VSTest
    testResultsFiles: '**/*.trx'

- task: DotNetCoreCLI@2
  inputs:
    command: publish
    publishWebProjects: True
    arguments: '--configuration $(BuildConfiguration)
      --output $(Build.ArtifactStagingDirectory)'
    zipAfterPublish: True

- task: AzureRmWebAppDeployment@3
  inputs:
    azureSubscription: 'YourSubscription'
    WebAppName: 'your-webapp'
    Package: $(Build.ArtifactStagingDirectory)/**/*.zip
    # Uncomment the attribute below if you are deploying to a Windows Web app
    # TakeAppOfflineFlag: true
```

and parallel jobs. Additional jobs and minutes can be purchased for a monthly fee. Open source projects receive multiple free parallel jobs. You can find more information about Azure Pipelines pricing at aka.ms/azure-pipelines-pricing.

Deploying your apps to Azure with Azure Pipelines allows you to create reusable CI/CD patterns and reliably automate each step of your deployments. You can create pipelines in multiple ways, use simple or complex workflows, and deploy securely to a wide range of deployment targets. Azure DevOps Projects provide automation for Azure Pipelines to help you quickly get started with deploying your apps to Azure services. YAML pipelines allow you to create configuration-as-code patterns that yield many benefits. To learn more, see aka.ms/sign-up-for-azure-pipelines, aka.ms/azure-devops-project and aka.ms/learn-azure-pipelines. ■

Micheal Learned *has spent more than 20 years working on software engineering projects inside and outside of Microsoft. Learned spends his time focused on family, friends, and all things DevOps and Cloud. You can reach him via Twitter: @mlhoop.*

Andy Lewis *writes about various things Azure DevOps, including version control and CI/CD pipelines. At Microsoft he's served customers of Windows, Office and Developer Division. Lewis has designed and developed content, multimedia, and apps for a wide range of audiences at IBM, Borland, Intuit, SAS Institute, and Microsoft.*

Thanks to the following Microsoft technical experts for reviewing this article: *Jason Conner, Matt Cooper*

7 Tips and Tricks for Azure App Service

Michael Crump

Azure Tips and Tricks (azuredev.tips) is a series that I created a year ago where I document my favorite secrets, shortcuts and handy features using Azure. I quickly came to appreciate the value of short and straight-to-the-point guidance for common scenarios that developers face every day. Over the last year I've gone from just a few posts to more than 150 tips spanning the entire Azure platform.

For this special edition of *MSDN Magazine* focused on the Microsoft Connect(); 2018 conference, I pulled together a collection of the seven most popular and valuable tips related to Azure App Service. These tips include working with files in the console, setting up staging environments and swapping between them, and routing traffic to different versions of your app to "Test in Production." I'll also cover how you can implement performance testing, adopt best practices for App Settings in Azure App Service, and clone a Web app that's especially helpful if you have customers all over the world.

This article discusses:

- Learn how to work with Deployment Slots in Azure App Service
- Learn how to perform load testing of your Web app in Azure App Service
- Discover the powerful logging and error diagnostic capabilities built into Azure App Service

Technologies discussed:

Azure Portal, Azure App Service, ASP.NET MVC

Working with Files in Azure App Service

In this tip I'll look at the files inside an Azure App Service Web site that allows you to easily manage your Web app environment without leaving the Azure Portal. To begin, go to the Azure Portal and select an App Service that you've already created and click on Console under Development Tools. This brings up a command prompt from where you can work with your Azure App Service (see **Figure 1**).

As you can see in **Figure 1**, the prompt starts in D:\home\site\wwwroot. From here I can enter "dir" to see a current directory listing, as shown in **Figure 2**.

I can perform basic commands here and use the command "type <filename>" to display the output of a file to the screen. You can create directories and rename files, but keep in mind that this is a sandbox environment. Some commands that require elevated permissions may not work from the Console inside of Azure App Service. For a list of available commands from the prompt, just type "help" in the console window.

Visual Studio Code Experience

There's another option under Development Tools in the Azure App Service Portal called App Service Editor. You can find this item located just two spots down from the Console item that I described in the previous tip. Click App Service Editor and select a file, and you'll see a screen similar to the image in **Figure 3**.

If you're familiar with Visual Studio Code, you'll find yourself right at home as you explore, search and manage Git. The App

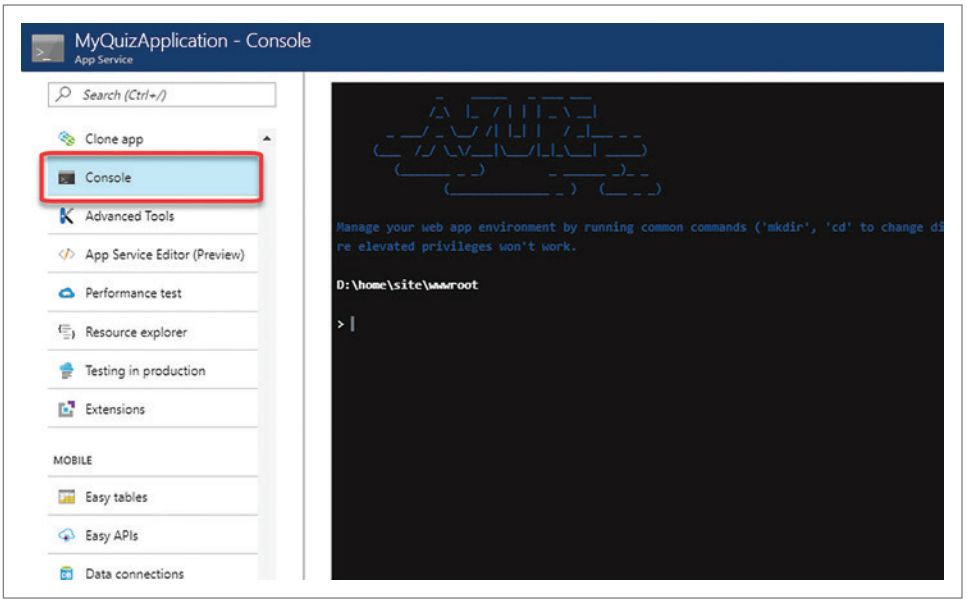


Figure 1 **The Console Inside Azure App Service**

Service Editor also makes it easy to add, edit, or delete files, and just like in Visual Studio Code, you can modify your settings and even change your theme. You can also take advantage of the output window to review the Application Logs associated with your app service.

Kudu Diagnostic Console

No App Service tutorial is complete without mentioning Kudu Diagnostic Console. You can access it from within the App Service Editor by going to your app and then clicking Open Kudu Console, or by looking for the Advanced Tools option under Development Tools.

To navigate or type in a command, just click on the folder name. You can easily manipulate files from here, but I personally like the App Service Editor better for that functionality. I primarily use the Kudu Diagnostic Console to easily discover and work with the REST API and work with the Process explorer.

Creating a Deployment Slot

Let's move on from the wonders of the App Service Editor to explore how to streamline testing of Web apps in production with Azure App Service. I'll start with a look at deployment slots, an enormously useful feature that lets you deploy different versions of your Web app to different URLs. You can then test a specific version of your app and swap content and configuration between slots.

Start by going to the Azure Portal and selecting an existing App Service that you have running and under Deployment click Deployment Slots. Then click on the Add Slots button. Enter a descriptive name such as staging, then use an existing configuration source, as shown in **Figure 4**. For this example I'll use a "production" Web app—the cool quiz application (github.com/mbcrump/jsQuizEngine) that's already deployed.

Once that's done, go back to Deployment Slots to see the status, which should read "Running." Click on the new staging site that was just created. You'll notice that it has appended the word staging. You'll also notice that there's a new site at yoursitename-staging.azurewebsites.net.

Let's say you need to push a new version of the existing quiz application to this staging slot. Go to Deployment Center and select External and enter the following URL into the text box: github.com/mbcrump/jsQuizEngine.git. Click OK. You may find that you need to click Sync to make the changes take effect. A check mark should appear once the process has successfully completed.

Wait a couple minutes for your code to be pulled down from Git, then go to the new URL of your site. You can find the URL on your overview page. In my case it's at myquizapplication-staging.azurewebsites.net. You should now see your new Web site, with jsQuizEngine version 2 displayed in large font.

> The benefit of this is that your users won't experience downtime while you tinker, and you can continue working in your preferred space until ready to move to production.

At this point, I could return to the original app service that I created and swap between the two sites that I now have. For example, I might want to move the staging site over to the production site, and vice versa. The benefit of this is that your users won't experience

Figure 2 **The wwwroot Directory Listing**

```
Volume in drive D is Windows
Volume Serial Number is FE33-4717

 Directory of D:\home\site\wwwroot

09/21/2017  08:35 PM    <DIR>          .
09/21/2017  08:35 PM    <DIR>          ..
09/20/2017  09:03 PM    <DIR>          css
09/20/2017  09:03 PM             5,351 Default.html
09/20/2017  09:03 PM    <DIR>          js
09/20/2017  09:03 PM             1,950 jsQuizEngine.sln
09/20/2017  09:03 PM               304 jsQuizEngine.userprefs
09/20/2017  09:03 PM            31,744 jsQuizEngine.v12.suo
09/20/2017  09:03 PM    <DIR>          PrecompiledWeb
09/20/2017  09:03 PM    <DIR>          quiz
               4 File(s)         39,349 bytes
               7 Dir(s)   1,072,893,952 bytes free
```

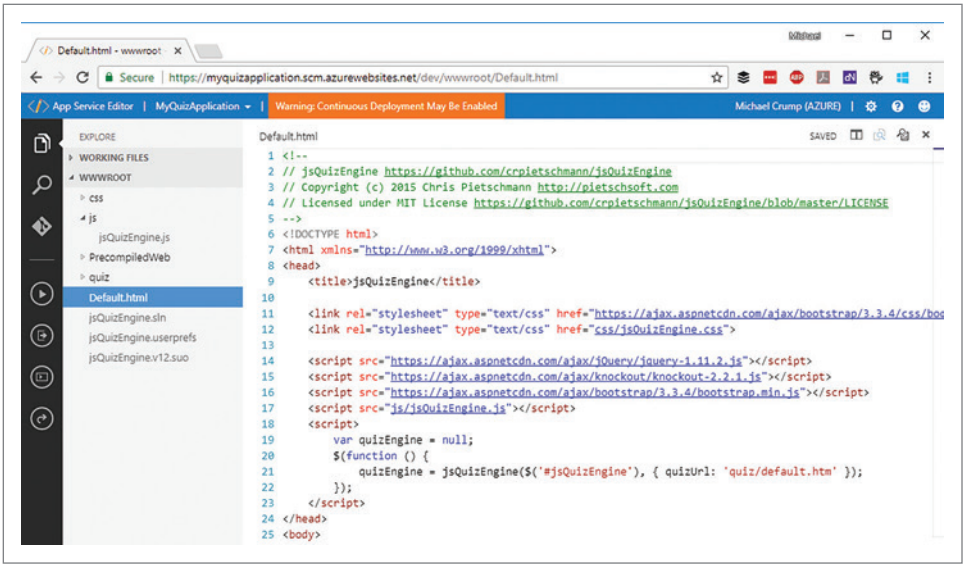


Figure 3 **The App Service Editor**

downtime while you tinker, and you can continue working in your preferred space until ready to move to production.

Test Web Apps in Production

In this tip, I'll look at a feature called Testing in Production, which does exactly what it says. Not scary at all! But hold up. If you're not familiar with deployment slots, you'll want to look at the previous tip before you dive in.

> What's Static Routing? It's a feature of deployment slots that lets you control how traffic is distributed between your production and other slots.

To get started with this feature, go to the Azure Portal and select App Service and under Development Tools click Testing in Production. The first thing you'll see is the title Static Routing and you'll notice two options that are for a deployment slot name and traffic percentage (to which you'll want to route).

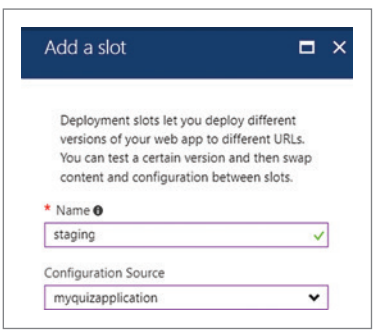


Figure 4 **Adding a Deployment Slot**

What is Static Routing? It's a feature of deployment slots that lets you control how traffic is distributed between your production and other slots. This is useful if you want to try out a new change with a reduced number of requests and then gradually increase the number of requests that get the new behavior.

You'll want to split the traffic to your site into two groups to test the new site and see if customers like it. Because this is just a demo, you'll want to send a large number of folks to the staging site as shown in **Figure 5**.

Now keep in mind that you have two versions of your site. One that's production and one in staging. They're identical except that the staging site includes text in a large font that reads "jsQuizEngine version 2." You don't want to swap these sites, rather you just want to distribute traffic between them.

You can test this by going to the production URL and constantly refreshing the site until the staging site is shown with the production URL.

What happens when the user leaves the site? By default, a cookie keeps track of the event. You can find this cookie yourself by bringing up the developer tools and looking for the cookie called "x-ms-routing-name."

You could actually force the old production site by setting the x-ms-routing-name cookie to self or providing it in the URL query string, such as http://myquizapplication.azurewebsites.net/?x-ms-routing-name=self. You could even use the URL to let your users test different versions of your site. For example, I could use http://myquizapplication.azurewebsites.net/?x-ms-routing-name=staging to let users try my new Web site before I push it live. This is very neat stuff, folks!

Load Test Web Apps

Next, let's look at a simple and quick way to perform load testing of Web apps. Load Testing allows you to gauge your Web app's performance and determine if it can handle increased traffic during peak times. Log into your Azure account and go to the App Service you created and look under Development Tools. You'll see Performance Test there.

> In this tip, I'll look at a feature called Testing in Production, which does exactly what it says.

Inside the blade, select New to see the Configure Test option, which you can leave at the default setting of Manual Test or change to Visual Studio Web Test. The main difference between the two: Visual Studio Web Test lets you select multiple URLs and even use

Visual Studio LIVE! TRAINING SEMINAR

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

ASP.NET Core in the Microsoft Cloud

Dallas, Texas

February 6 - 7, 2018

Two Hot Topics in Two Days

> February 6: ASP.NET Core with Azure DevOps

> February 7: ASP.NET Core Microservices with Docker, K8s, and AKS

TOP REASONS TO JOIN US IN DALLAS

- In-Depth Training
- Two Hot Topics in Two Days
- Networking
- Flexibility

REGISTER TODAY!

Use Promo Code MSDN

VSLIVE.COM/DALLAS

an HTTP Archive file (such as one created by Fiddler). For this walkthrough, use the setting Manual Test and select a name and location, making sure you leave the defaults at 250 users for five minutes.

Imagine this use-case scenario. You have a Web app presenting an item for sale, with an upcoming promotion that last year resulted in 175 users being connected for five minutes. Users complained then that the site was slow, so you're anxious to improve customer satisfaction by reducing the page load time. To help do this, you test your Web app with a heightened load of 250 (or even 2,500) users for five minutes.

Run the test and you'll be presented with data such as successful and failed requests, performance under load, and more. For the scenario I ran, the average response time decreased, while requests per second increased. If I had chosen a larger time window, I'd probably have more detail to help locate where I could improve the performance of the Web app. You may also want to check out the Status Message panel for additional information about the test.

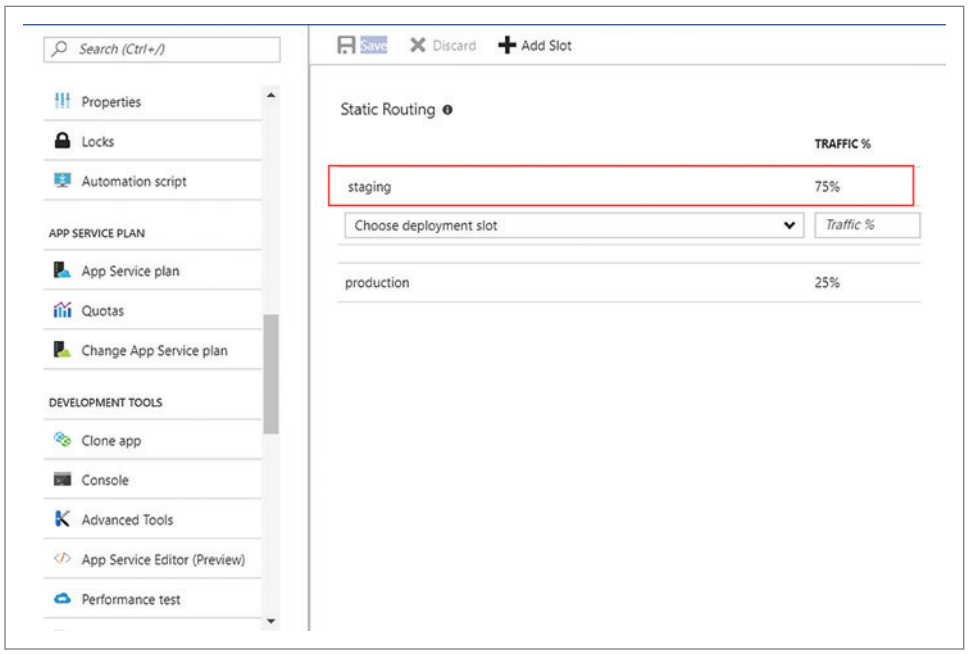


Figure 5 **Adjusting the Static Routing Traffic Percentage for the Staging Site**

> A general rule of thumb is to use Connection Strings for database connection strings and App Settings for key-value pair application settings.

I was able to do this without writing code and with just a couple of clicks in the portal. Just keep in mind that there is a charge for performing a load test in terms of virtual users.

Working with App Settings

Here's a trick to take advantage of App Settings to store a Key/Value pair securely in Azure and access it in your Web app.

App Settings are used to store configurable items without making any changes to the code. The key-value pairs are stored behind the scenes in a configuration store, which is nice because sensitive information never shows up in a web.config file or wherever else you store secrets. In order to take advantage of this, you must log into your Azure account and go to the App Service you created. Look under Settings to see Application Settings.

Let's add an App Setting in Azure, with the key of Environment and the value set to Staging. Open or create an ASP.NET MVC app (for example) and modify the appSettings section of the web.config file to add the Environment key/value pair as shown here:

```
<appSettings>
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="Environment" value="Production" />
</appSettings>
```

Now it's in the Web configuration section of the app. In order to see the value, you need to add it to a page inside of your app to display the value for this example.

Using ASP.NET MVC, navigate to your appname/Views/Home/Index.cshtml file and add the following using statement followed by a call to ConfigurationManager:

```
@using System.Configuration
@{
  ViewBag.Title = "Home Page";
}

<div class="jumbotron)
  <h1>Testing App Settings</h1>
  @ConfigurationManager.AppSettings["Environment"]
</div>
```

If you run the application locally, you'll see Production as it's coming from the web.config file, but if you run it inside Azure, you'll see Staging as it's coming from the App Settings configuration store located in Azure. Neat stuff!

You may have noticed Connection Strings visible right below the App Settings option and wondered when to use it. A general rule of thumb is to use Connection Strings for database connection strings and App Settings for key-value pair application settings. If you examine your web.config file, you'll see that there's also a section for connectionStrings, just as there's a section for appSettings.

Cloning Web Apps

Cloning is the ability to clone an existing Web App to a newly created app that's often in a different region. This enables customers to deploy a number of apps across different regions quickly and easily.

For instance, a company might have an existing Web app in the West U.S. region, and would like to clone the app to the East U.S. region to reduce latency and improve performance for users in that area.

To do this, log into your Azure account and go to the App Service you created and under Development Tools click Clone App. Enter the following information:

- App Name: Make sure to use something unique as this site will live in something.azurewebsites.net
- Resource Group: Create a new group or use an existing one
- App Service Plan/Location: This is a good time to associate a new plan that will determine the location, features, cost and compute resources associated with your app

Besides changing the location, this is also a great time to determine the required app service plan. You might not need all the horsepower to serve this site if you expect very low traffic in that region.

- Clone Settings: This will copy the content and certificates of your app into a newly created application—you can also copy things like App Settings, Connection Strings, Deployment Source and Custom Domains
- Application Insights: Enable this service to help you detect and diagnose issues and more with .NET apps

Finally, there's Automation Options, which brings you to the Azure Resource Manager templates that are so valuable. Learn more about Azure Resource Manager templates at bit.ly/2RTVfri.

> You can easily change options such as the physical disk size of log files that Azure will store, as well as the number of days to keep the log files in retention.

Azure Resource Manager enables you to work with the resources in your solution as a group. You can deploy, update or delete all the resources for your solution in a single, coordinated operation. You use a template for deployment, and that template can work for different environments such as testing, staging and production. Resource Manager provides security, auditing and tagging features to help you manage your resources after deployment. Read more at bit.ly/2Pv9aY5.

Once everything is set up, press Create and you'll see Deployment in Progress overlay on the screen. You can click on it while deploying to see details such as status, duration of deployment, location and more.

Working with Log Stream

Log Stream is the ability to see logging information in real time (or as close to it as possible). You can do this using the Azure Portal or through some of the CLI tooling such as PowerShell or BASH. To take advantage of this, log into your Azure account, go to your App Service and open the Log Stream item under Monitoring. You'll see that most of the Application logs are switched off. To turn them on, you'll need to go to the Diagnostic logs setting in the same panel.

Here you'll see the following options:

- **Application Logging** allows you to collect diagnostic traces from your Web code. This is required for the streaming log feature and turns itself off after 12 hours.
- **Application Logging (Blob)** produces logs that are collected in the blob container specified under Storage Settings.
- **Web server logging** gathers diagnostic information for your Web server.
- **Detailed error messages** gathers detailed error messages from your Web app.
- **Failed request tracing** gathers diagnostic information on failed requests.

You can easily change options such as the physical disk size of log files that Azure will store, as well as the number of days to keep the log files in retention. You can also download the log files via FTP and FTPS.

Go ahead and turn Application Logging(Filesystem) to On and the Level to Info and press save. Go back and click on the Log Stream setting and visit your Web page (that's hosted on *.azurewebsites.net). You should see a log. If you look at the first couple of lines, you'll see a ReadyForRequest on port 80, followed by a request from our Web browser that includes pulling down the favicon.ico file (which is the only image being sent down).

Add Trace Logging in Code

Using the ASP.NET MVC app from earlier, navigate to your appname/Controllers/HomeController.cs file (or wherever you'd like to test this functionality) and add the following line:

```
System.Diagnostics.Trace.WriteLine("Entering the About View");
```

Head back over to Diagnostic logs and ensure the Application Logging (Filesystem) Level is set to Verbose, then press save since we just used a WriteLine command. Switch back over to the Log Stream option and trigger the error by opening the site and navigating to the About page. Review the Application Log after it calls your Trace.WriteLine command. You should see the message "Entering the About View" that was specified in the Trace.WriteLine call.

Wrapping Up

Today you learned a number of tricks in Azure App Service that can help you be more productive. But this article barely scratches the surface of the things you can do. Over the past year I've compiled a library of valuable tips, tricks and secrets that cover such things as using .NET Core, Application Insights for rich analytics, user authentication and more. If you liked what you saw in this article, go check out the entire collection of Azure App Service tips at azuredev.tips, or visit videos.azuredev.tips to view short videos of many of these tips and tricks. ■

Michael Crump *works at Microsoft on the Azure platform and is a coder, blogger and international speaker of various cloud development topics. He's passionate about helping developers understand the benefits of the cloud in a no-nonsense way. You can reach him on Twitter: @mbcrump or by following his blog at michaelcrump.net.*

Thanks to the following Microsoft technical expert for reviewing this article: *Cecil Phillip*

Exploring the Xamarin.Forms Shell

David Ortinau

Xamarin.Forms is a favored toolkit for cross-platform developers who love XAML and C#, because it maximizes code sharing while also providing full access to all the native platform APIs and UI controls. This capability comprises technologies and concepts that can be both exhilarating and confusing when you're getting started. The truth is that some developers find it frustrating at the outset. You chose Xamarin to be productive, and the last thing you want to encounter is unwanted hassle. This year at Connect(); we're thrilled to introduce Xamarin.Forms Shell, a new default starting point for mobile application development that reduces complexity and increases productivity.

As the name suggests, Shell is fundamentally a container that takes care of the basic UI features that every application needs, so you can focus on the core work of your application. Existing iOS and Android applications can also easily adopt Shell and benefit immediately from the improvements to navigation, UI performance and extensibility. Shell offers the following benefits:

- A single place to describe the visual structure of applications
- A common navigation UI and omnipresent navigation service with deep linking
- An integrated search handler to improve the overall in-app search experience
- An extensible-by-default philosophy to add more versatility and flexibility

This article discusses:

- How to describe the visual structure of your application in a single file
- Enabling navigation via URI routing
- Using a flyout, bottom tabs and top tabs
- Styling and customizing via templates

Technologies discussed:

Xamarin.Forms Shell

The App

At the beginning of every project, someone sketches out the structure of the application for you to build (and hopefully not just in their head). Sometimes it's provided in a design comp, and sometimes it's just penciled in on paper. Shell makes it super easy to take that content and translate it into a running application container that's ready for anyone to populate with content and functionality.

In this article, I'll use the example of a mobile shopping application called Tailwind Traders. This is a new reference application the team has made to demonstrate how you can use Xamarin.Forms Shell, Azure, Cognitive Services, and several other features and services. Take a look at the design comps provided by our awesome design team, shown in **Figure 1**.

As you can see from the displayed screens, the app provides all the obvious functionality you need, including login and registration flow, a browsing experience with product categories and search, and a checkout flow. This app also leverages the device camera and the power of the Azure Custom Vision API to identify products in real time.

The Quick Start

Let's quickly scaffold out this application using Shell. Open Visual Studio 2019 and start a new cross-platform application with Xamarin.Forms. For the purpose of this article and to understand the power of Shell, let's start with a Blank project and build up the structure of the Tailwind Traders app.

Once the project files are generated, open the App.xaml.cs and notice that MainPage is set to a new Shell instance. (You can download Shell templates from aka.ms/xf-shell-templates.) Structurally, this

is the only difference from a typical Xamarin.Forms application that you may have seen in the past. Here's the code:

```
namespace TailwindTraders.Mobile
{
  public partial class App
  {
    public App()
    {
      InitializeComponent();

      MainPage = new AppShell();
    }
  }
}
```

Open the AppShell.xaml in the root of your .NET Standard library project, as shown in **Figure 2**.

Let's break down the pieces of this file. A Shell consists of three hierarchical elements: ShellItems, ShellSections and ShellContent. Every ShellContent is a child of a ShellSection, which is a child of a ShellItem—all parts of the Shell. None of these by themselves represent the UI, but rather the organization of your application's architecture. Shell takes these items and produces the appropriate navigation UI for the platform it's running on:

- **ShellItem:** The top-level structure of the application represented by an item in the flyout. Can contain multiple ShellSections.
- **ShellSection:** A grouping of the application content, which is navigable by bottom tabs. Can contain one or more ShellContents, with multiple ShellContents navigable by top tabs.
- **ShellContent:** The ContentPages of your application.

I can use these three elements to describe the visual structure of the Tailwind Traders mobile application. Ignoring the login and registration flow, I'll add several ShellItems to host content, represented with a flyout menu on the left.

Why not use names like FlyoutItem, BottomTab, TopTab for Shell concepts? Our team at Microsoft has had many discussions about this and feels that Xamarin.Forms caters to known and future platforms that sometimes don't share the exact concepts of tabs or menus. By keeping the nomenclature abstract, we let you decide through styling and templates if these elements should be represented consistently across divergent platforms, or if they should adhere to each platform's design aesthetic. Of course, your feedback is always welcome in these matters!

Figure 3 provides an example. Here you see a menu in the flyout (the lower two-thirds of the UI), which is populated automatically by ShellItems. This allows you to navigate to the different areas of your application. In addition to those items, you can explicitly add menu items that aren't associated with a ShellItem. The flyout header at the top (the two buttons) presents special content that consists of anything you wish to present in that space. To declare a custom FlyoutHeader within the Shell.xaml, use this code:

```
<Shell.FlyoutHeader><local:FlyoutHeader />   </Shell.FlyoutHeader>
```

The header element allows you to control how it behaves when users scroll the display. There are three options:

Figure 1 **Design Comps for Tailwind Traders Sample App**

- **Fixed:** The header remains fixed while the content below scrolls.
- **Scroll:** Scrolls with the menu items.
- **CollapseOnScroll:** Collapses in a parallax fashion as you scroll.

To adjust this behavior, set the FlyoutHeaderBehavior property on your Shell to the desired value just previously detailed. For now, we'll keep it fixed with the following code:

```
<Shell
  x:Class="TailwindTraders.Mobile.Features.Shell.Shell"
  FlyoutHeaderBehavior="Fixed"
  ...
  >
  ...
</Shell>
```

Next, let's set up the content. Looking at the design, I can see that there's a home screen, a series of product categories, a profile and, finally, a logout screen. Let's start with the home screen, with the following XAML code:

```
<ShellItem Title="Home">
  <ShellSection>
    <ShellContent>
      <local:HomePage />
    </ShellContent>
  </ShellSection>
</ShellItem>
```

Figure 2 **A Single Page Shell.xaml**

```
<?xml version="1.0" encoding="UTF-8"?>
<Shell xmlns="http://xamarin.com/schemas/2014/forms"
       xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
       xmlns:local="clr-namespace:TailwindTraders"
       RouteHost="tailwindtraders.com"
       RouteScheme="app"
       FlyoutBehavior="Disabled"
       Title="TailwindTraders"
       x:Class=" TailwindTraders.AppShell">

  <ShellItem>
    <ShellSection>
      <ShellContent>
        <local:MainPage/>
      </ShellContent>
    </ShellSection>
  <ShellItem>

</Shell>
```

Breaking down this XAML from the inside out, I've added the HomePage to the app, which will be the first ContentPage to launch, because it's the first content declared in the shell file. This is the same ContentPage type you use in your existing Xamarin.Forms applications, now hosted within a Shell context.

For this design I only need to set a title, but ShellItem also provides the FlyoutIcon property, which lets you provide an image to display on the left of the item. Icons may be any Xamarin.Forms ImageSource.

Go ahead and run the app. On the homepage click the hamburger icon to open the flyout menu. Tapping this menu item navigates you to the home screen (which is currently the only screen). Let's get to work adding more to that.

Next, I'll implement the product categories, such as "Holiday decorations," "Appliances" and the like. I could add ShellItems for each of them, but because the product category pages are all the same page with different content, I can be clever about it. I'll use a simple MenuItem to navigate to the same page and pass data via the CommandParameter to avoid unnecessary duplication of pages. Here's the code to add a MenuItem in the Shell.xaml:

```
<Shell.MenuItems>
  <MenuItem
    Command="{Binding ProductTypeCommand}"
    CommandParameter="1"
    Text="Holiday decorations" />
</Shell.MenuItems>
```

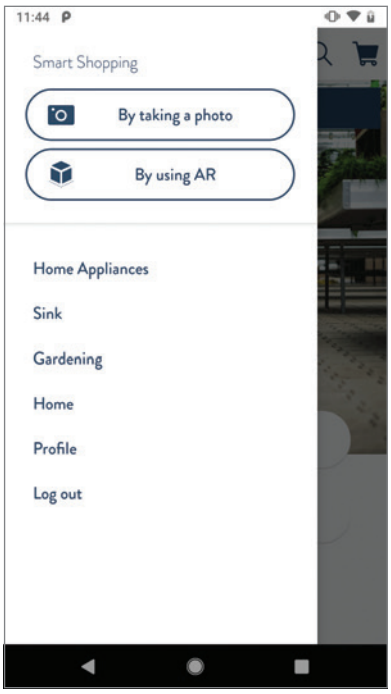


Figure 3 **Elements of the FlyoutMenu**

A great feature of Shell is that it supports data binding. In this case I have a "Command" on a view model that can execute the navigation. Just like ShellItems, MenuItems take text and an icon. Also, just like ShellItems, I can provide styling or even a custom template to further customize the design by setting the MenuItemTemplate property on the Shell.

I can add more menu items for each category to complete the task. **Figure 4** shows the code for all the menu items, while **Figure 5** shows the visual result in the flyout menu of the app.

Adding More Pages

Let's now add the Profile page from the design to our app. Add a new ContentPage to the project and then move back to the Shell.xaml file. You could copy the XAML (shown in **Figure 4**) used for the HomePage and just replace it with the Profile page, but you risk bogging down the application because HomePage is created immediately during the application startup. To avoid having all the pages of the application loading at once, I use a data template, like so:

```
<ShellContent
  Title="Profile"
  ContentTemplate="{DataTemplate local:ProfilePage}" />
```

Rather than providing the ContentPage directly to the content property of the ShellContent, I supply a data template. When the user navigates to a screen, the Shell instantiates the requested page dynamically.

One thing to note with this treatment compared to the HomePage is that I've omitted the ShellItem and ShellSection wrappers and

Figure 4 **All the Menu Items**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
  x:Class="TailwindTraders.AppShell"
  FlyoutHeaderBehavior="Fixed"
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:TailwindTraders.Views"
  Title="Tailwind Traders"
  x:Name="theShell"
  Route="tailwindtraders"
  RouteHost="microsoft.com"
  RouteScheme="app">

  <Shell.MenuItems>
    <MenuItem
      Command="{Binding ProductTypeCommand}"
      CommandParameter="1"
      Text="Holiday decorations" />

    <MenuItem
      Command="{Binding ProductTypeCommand}"
      CommandParameter="2"
      Text="Appliances" />

    <MenuItem
      Command="{Binding ProductTypeCommand}"
      CommandParameter="3"
      Text="Bathrooms" />

    <MenuItem
      Command="{Binding ProductTypeCommand}"
      CommandParameter="4"
      Text="Doors &amp;amp; Windows" />

    <MenuItem
      Command="{Binding ProductTypeCommand}"
      CommandParameter="5"
      Text="Flooring" />

    <MenuItem
      Command="{Binding ProductTypeCommand}"
      CommandParameter="6"
      Text="Kitchen" />

    <MenuItem
      Command="{Binding ProductTypeCommand}"
      CommandParameter="7"
      Text="Storage" />
  </Shell.MenuItems>

  <ShellItem Title="Home">
    <ShellSection>
      <ShellContent>
        <local:HomePage />
      </ShellContent>
    </ShellSection>
  </ShellItem>

</Shell>
```

Spreadsheets Everywhere.

SpreadsheetGear 2017 Released

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

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

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

Scalable Reporting

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

WIN Windows Forms | Silverlight | WPF

Powerful Controls

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

Comprehensive Charting

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

Fastest Calculations

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

MARCH 3-8, 2019 ➤ BALLY'S HOTEL AND CASINO

AN OASIS OF EDUCATION DAZZLING IN THE DESERT

Las Vegas

INTENSE DEVELOPER TRAINING CONFERENCE

In-depth Technical Content On:

- AI, Data and Machine Learning
- Cloud, Containers and Microservices
- Delivery and Deployment
- Developing New Experiences
- DevOps in the Spotlight
- Full Stack Web Development
- .NET Core and More

AGENDA AVAILABLE AT VSLIVE.COM/LASVEGAS

New This Year!

On-Demand Session Recordings Now Available

Get on-demand access for one full year to all keynotes and sessions from Visual Studio Live! Las Vegas, including everything Tuesday – Thursday at the conference.

SUPPORTED BY

Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360

#VSLIVE

HEAR FROM YOUR PEERS!

See what past attendees had to say about Visual Studio Live!

CHIRANJEEVI ASHOK PUVVULA

Software Engineer, Ascend Learning

"This is my first time attending a VSLive! Event. I love it. I learned about a lot of new technologies in the first few days. I plan to use these new technologies to create what I want. I'll definitely be back next year!"

KATIE GRAY

Senior Lecturer, The University of Texas at Austin

"I have loved learning about all of the latest offerings in the Visual Studio ecosystem from experts in the field. All of the sessions provided candid information about what is new and what is coming. It has also been great to network with fellow developers to hear about what they are working on at their jobs."

JULIAN PIANE

Senior Application Developer, The Archer Group

"My favorite part of VSLive! was having the ability to have in-person conversations with MVP's and fellow developers about not only hard tech, but also about the industry as a whole."

Register by January 11 & Save Up To $400!

Use Promo Code MSDN

Your Adventure STARTS HERE!

vslive.com/lasvegas

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

placed the title directly in the ShellContent. This is much less verbose and the Shell knows how to handle it for me by supplying the required logical wrappers. It's also important to note that these wrappers don't introduce UI views to the tree. Shell is written with rendering speed and memory consumption in mind. The result of this is that the performance impact on the Android OS is kept low by hosting the same content and UI you currently have within this new Shell context. Of course, you're still ultimately responsible for architecting the internals of your applications, but Shell offers a great starting point.

Styling the Flyout

You can style aspects of Shell and the FlyoutMenu just as you would any other XAML element using CSS or XAML styling. What if you wish to go further with how the flyout menu item appears? Looking back at the design in **Figure 3**, the menu items are bolder than the other shell items.

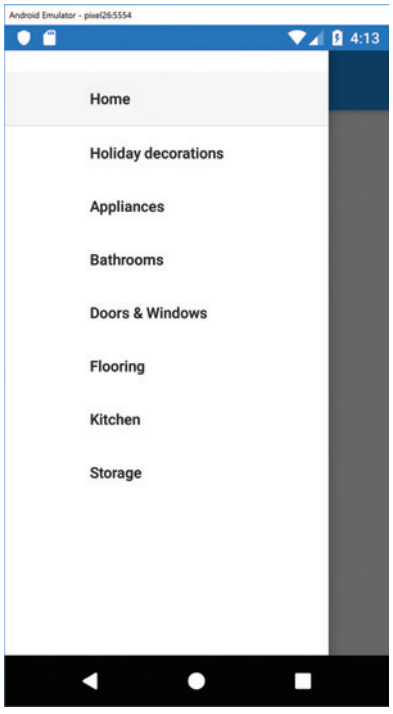


Figure 5 **Flyout with All Menu Items**

The display of menu items and shell items are extensible by providing a DataTemplate to Shell. A MenuItem is rendered in the flyout menu using the Shell's MenuItemTemplate, and a ShellItem is rendered using an ItemTemplate. To take full control of how these look, set each property to a DataTemplate containing a custom ContentView. Shell will provide the Title and Icon bindable properties to the template BindingContext, as shown in **Figure 6**. You can see the visual result in **Figure 7**.

In addition to customizing the item renderers, let's add a nice header to the flyout that includes a box with a label and two buttons for quick access to camera features. As with the other templates, add one for the FlyoutHeaderTemplate in the Shell.xaml file. The content can be any ContentView, so here use a StackLayout to vertically position the child controls, as shown in the code in **Figure 8**. Add some styling to get it close to the design comp, and run the app to see the result as shown in **Figure 9**. You can set FlyoutHeaderBehavior in the Shell element to determine if the header is fixed, or if it can scroll or collapse when the user scrolls the screen.

Figure 6 **Customizing the Item Template for ShellItems in Shell.xaml**

```
<Shell.ItemTemplate>
  <DataTemplate>
    <ContentView HeightRequest="32">
      <ContentView.Padding>
        <Thickness
          Left="32"
          Top="16" />
      </ContentView.Padding>
      <Label Text="{Binding Title}" />
    </ContentView>
  </DataTemplate>
</Shell.ItemTemplate>
<Shell.MenuItemTemplate>
  <DataTemplate>
    <ContentView HeightRequest="32">
      <ContentView.Padding>
        <Thickness
          Left="32"
          Top="16" />
      </ContentView.Padding>
      <Label Text="{Binding Text}" FontAttributes="Bold" />
    </ContentView>
  </DataTemplate>
</Shell.MenuItemTemplate>
```

Navigation

Now it's time to implement the command that navigates to the menu item pages. To do so, I'll use the new URI-based routing that Shell introduces. URIs let users jump instantly to any part of the application, and even provide the ability to go backward without having to create all the pages between the two points. Let's look at how this is accomplished.

First, I need to declare the routes, beginning with the scheme and host for my app, like so:

```
<Shell
  Route="tailwindtraders"
  RouteHost="www.microsoft.com"
  RouteScheme="app"
```

Putting these pieces together into a URL I end up with the following URI: app://www.microsoft.com/tailwindtraders.

Each Shell element I've defined in the Shell file also takes a route property, which I can later use to navigate programmatically. For pages that aren't represented by a Shell element I can explicitly register a route. This is what I'll do for those menu items added to the flyout. Each of them will navigate to a ProductCategoryPage, a page that displays a list of products for a specific category. Here's the route registration code:

```
Routing.RegisterRoute("productcategory", typeof(ProductCategoryPage));
```

Now I can declare the necessary routes in the constructor of the Shell.cs, or anywhere that runs before the routes are called. Menu items expose a command to implement the necessary navigation, as you can see in this code:

```
public ICommand ProductTypeCommand { get; } =
  new Command<string>(NavigateToProductType);

private static void NavigateToProductType(string typeId)
  {
    (App.Current.MainPage as Xamarin.Forms.Shell).GoToAsync(
      $"app:///tailwindtraders/productcategory?id={typeId}", true);
  }
```

Another great benefit of Shell is that it has static navigation methods that are accessible from anywhere in the application. Gone are the days of worrying if the navigation service was available, passing it around from view to view models and adding navigation pages to wrap everything. Now, you can grab a reference to the application Shell, which is the MainPage of your application, accessible as a property of App.Current. You can see this in the previous code snippet. To execute the navigation, call the GoToAsync method, passing in a valid URL as a ShellNavigationState. A ShellNavigationState may be constructed from a string or a URI. Look at the code again, and you can see that GoToAsync also allows you to supply only a string, and Shell will do the work to instantiate a ShellNavigationState.

Data can be passed between views and view models with querystring parameters. Shell will set those values directly on the ContentPage or ViewModel when you decorate the appropriate properties with query property attributes, as shown in **Figure 10**.

The QueryProperty takes the public property name ("TypeID" in this example) from your receiving class and the querystring parameter name ("id" in this example) used in the URL.

Intercepting the Back Action

Intercepting Back action is a common requirement in mobile app development, and it can be a challenge with Xamarin.Forms. Shell remedies this problem, by letting you hook into the navigation routing before and after it's completed to implement a myriad of custom needs. Here's an example of navigation handling, first with XAML code assigning an event handler:

```
<Shell    ...
  Navigating="Shell_Navigating"
```

And then the C# code for the event handler:

```
private void Shell_Navigating(object sender,
ShellNavigatingEventArgs e)
{
  if (// Some Boolean evaluation)
  {
    e.Cancel(); // Do not allow this navigation AND/OR do something else
  }
}
```

On an instance of Shell, add an event handler to the Navigating event. In your codebehind, the ShellNavigatingEventArgs provides the base details of the navigation, as shown in **Figure 11**.

Tabs, Tabs, Everywhere Tabs

The flyout menu is a popular UI pattern for navigation. As you think about the hierarchy of content within your application, the top or outermost level of navigation is the flyout menu. From there, the next level of detail is the bottom tab. When you don't have a flyout present, the bottom tabs are generally considered to be the top level of navigation in an application. Then within the bottom tabs, the next level of navigation would be the top tabs. Beyond that you're into single pages that push one to another. This is an opinionated approach that Shell takes to providing navigation UI.

Let's start with bottom tabs. Each ShellSection within a single ShellItem can be represented as a bottom tab when there's more than one. Here's an example of XAML code producing bottom tabs for an app:

```
<ShellItem Title="Bottom Tab Sample" Style="{StaticResource BaseStyle}">
  <ShellSection Title="AR" Icon="ia.png">
    <ShellContent ContentTemplate="{DataTemplate local:ARPage}"/>
  </ShellSection>
  <ShellSection Title="Photo" Icon="photo.png">
    <ShellContent ContentTemplate="{DataTemplate local:PhotoPage}"/>
  </ShellSection>
</ShellItem>
```

This code presents two ShellSections in a single ShellItem. These ShellSections are represented in the UI as tabs at the bottom of the screen. What about when you don't need the flyout?

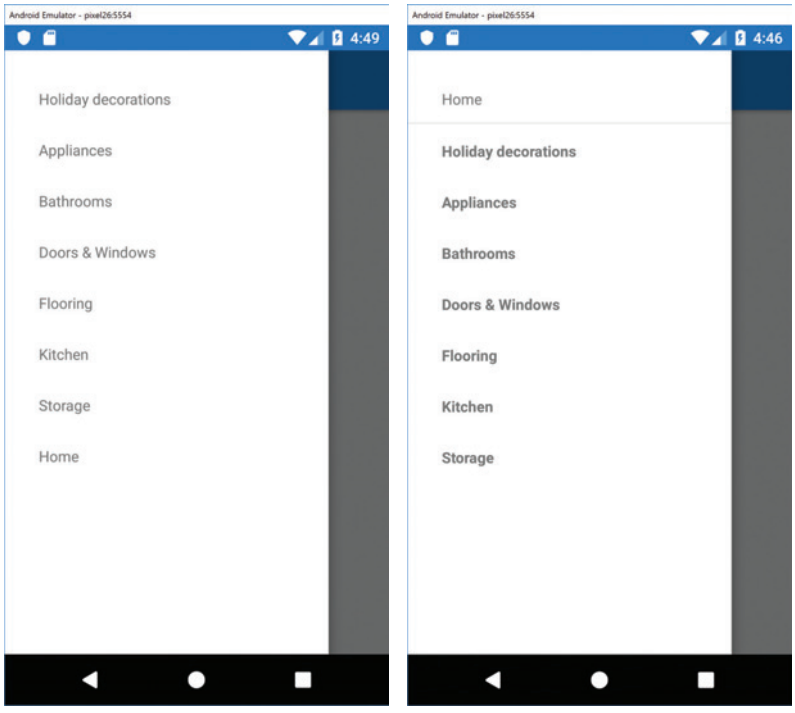


Figure 7 **Images of Flyout Item Template Results**

When there's only one ShellItem, it can be hidden altogether by setting the FlyoutBehavior to Disabled. Tabs can be styled using the existing style options or by supplying a custom renderer. Unlike the flyout menu items that can be customized data templates, the tabs are much more platform-specific. To style the

Figure 8 **FlyoutHeaderTemplate**

```
<Shell.FlyoutHeaderTemplate>
  <DataTemplate>
    <StackLayout HorizontalOptions="Fill" VerticalOptions="Fill"
      BackgroundColor="White" Padding="16">
      <StackLayout.Resources>
        <Style TargetType="Button">
          <Setter Property="BackgroundColor" Value="White" />
          <Setter Property="BorderColor" Value="#2F4B66" />
          <Setter Property="BorderWidth">2</Setter>
          <Setter Property="CornerRadius">28</Setter>
          <Setter Property="HeightRequest">56</Setter>
          <Setter Property="Padding">
            <Thickness
              Left="24"
              Right="24" />
          </Setter>
        </Style>
      </StackLayout.Resources>
      <Label FontSize="Medium" Text="Smart Shopping">
        <Label.Margin>
          <Thickness Left="8" />
        </Label.Margin>
      </Label>
      <Button Image="photo" Text="By taking a photo">
        <Button.Margin>
          <Thickness Top="16" />
        </Button.Margin>
      </Button>
      <Button Image="ia" Text="By using AR">
        <Button.Margin>
          <Thickness Top="8" />
        </Button.Margin>
      </Button>
    </StackLayout>
  </DataTemplate>
</Shell.FlyoutHeaderTemplate>
```

color of the tabs, use the style properties of the Shell class for TabBar items, like so:

```
<Style x:Key="BaseStyle" TargetType="Element">
  <Setter Property=
    "Shell.ShellTabBarBackgroundColor"
    Value="#3498DB" />
  <Setter Property=
    "Shell.ShellTabBarTitleColor"
    Value="White" />
  <Setter Property=
    "Shell.ShellTabBarUnselectedColor"
    Value="#B4FFFFFF" />
</Style>
```

Assigning the style class to the ShellItem applies those colors to all tabs in that section.

Now let's move on to top tabs. To have content navigable from top tabs, add multiple ShellContent items within a single ShellSection. Styling is applied just like the previous example for the bottom tabs. Here's the code:

```
<ShellItem Title="Store Home" Shell.
TitleView="Store Home"
  Style="{StaticResource BaseStyle}">
    <ShellSection Title="Browse Product">
      <ShellContent Title="Featured"
        ContentTemplate=
        "{DataTemplate local:FeaturedPage}" />
      <ShellContent Title="On Sale"
        ContentTemplate=
        "{DataTemplate local:SalePage}" />
    </ShellSection>
</ShellItem>
```

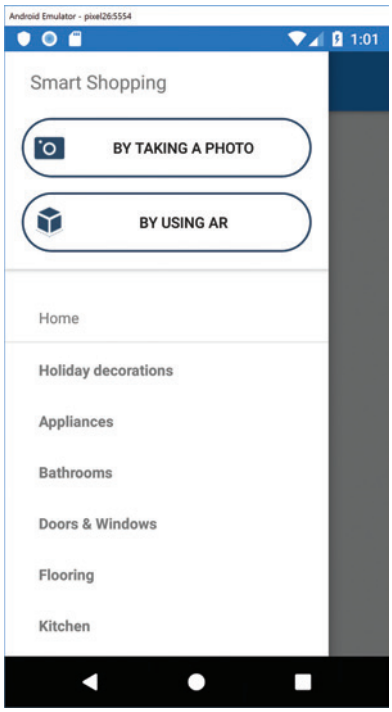


Figure 9 **Image of Flyout with Header**

The Roadmap

There's plenty more to discover in Xamarin.Forms Shell. I could continue on describing how to customize the navigation bar, the back button, and all about the very powerful search handler that makes it easier than ever to add search to a page. Those features and more are available now, and will be documented as we approach stable release.

The Shell journey is just starting. We hear loud and clear from Xamarin.Forms developers that you often need to make your iOS and Android applications look mostly or exactly the same. To address this, we plan to release Material Shell, which is an implementation of Shell that applies Google's Material Design styling as the starting point for all supported controls. The controls are still native so there are no performance or feature compromises.

Navigation transitions and segues are also on the way. With transitions you can control how one page animates to another (left-to-right, right-to-left, crossfade, curl and more). Segues are a declarative way to say, "When this button action happens, execute this route." It reduces the need to write GoToAsync navigation code and expresses with more clarity in XAML how things are connected. Blending transitions and Material Shell together we can provide some additional animations, such as the Hero animation where an element such as an image icon transitions from one page seamlessly to another page.

Figure 10 **Example of Query Attribute**

```
[Preserve]
[QueryProperty("TypeID", "id")]
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class ProductCategoryPage : ContentPage
{
  private string _typeId;

  public ProductCategoryPage()
  {
    InitializeComponent();

    BindingContext = new ProductCategoryViewModel();
  }

  public string TypeID
  {
    get => _typeId;
    set => MyLabel.Text = value;
  }
}
```

Figure 11 **ShellNavigatingEventArgs**

Element	Type	Description
Current	ShellNavigationState	The URI of the current page.
Source	ShellNavigatinState	The URI representing where the navigation originated.
Target	ShellNavigationState	The URI representing the navigation destined.
CanCancel	Boolean	Property indicating if it's possible to cancel the navigation.
Cancel	Boolean	Method to cancel the requested navigation.
Canceled	Boolean	Property indicating if the current navigation was canceled.

Start Exploring Today

Xamarin.Forms Shell is available today in the preview of Xamarin.Forms 4.0, which includes amazing new features like CollectionView, CarouselView and the all-new Material Visual that makes it easier than ever to start your Xamarin.Forms applications from a consistent, common UI styling point instead of the platform-specific blank point. Use your Visual Studio NuGet package manager to update to version 4.0-pre1 by toggling the pre-release option.

To make things even easier, we've created an updated package of project templates that unify on Shell and provide version 4.0-pre by default. Download and install the templates from aka.ms/xf-shell-templates. After you've done that, new Shell-powered templates will be available when you create a new Xamarin.Forms project.

As the Visual Studio 2019 pre-release continues to evolve, so will Xamarin.Forms 4.0 and Shell. We need your feedback. Let us know your thoughts and experiences by visiting aka.ms/xf-4-feedback. ■

David Ortinau *is a senior program manager for Mobile Developer Tools at Microsoft, focused on Xamarin.Forms. A .NET developer since 2002, and versed in a range of programming languages, Ortinau has developed Web, environmental and mobile experiences for a wide variety of industries. After several successes with tech startups and running his own software company, Ortinau joined Microsoft to follow his passion: crafting tools that help developers create better app experiences. When not at a computer or with his family, he's galloping through the woods.*

Thanks to the following Microsoft technical experts for reviewing this article: *David Britch, Jason Smith*

#VSLive

Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

April 22-26, 2019 | Hyatt Regency New Orleans

Spice Up Your Coding Skills in the Bayou

New Orleans

Intense Developer Training Conference

In-depth Technical Content On:

- AI, Data and Machine Learning
- Cloud, Containers and Microservices
- Delivery and Deployment
- Developing New Experiences
- DevOps in the Spotlight
- Full Stack Web Development
- .NET Core and More

New This Year!

On-Demand Session Recordings Now Available

Get on-demand access for one full year to all keynotes and sessions from Visual Studio Live! New Orleans, including everything Tuesday – Thursday at the conference.

Register by February 22 & Save Up To $400!

YOUR ADVENTURE STARTS HERE

vslive.com/neworleans

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

CONVERGE 360 an 1105 MEDIA company