THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2019 VOL 34 NO 7

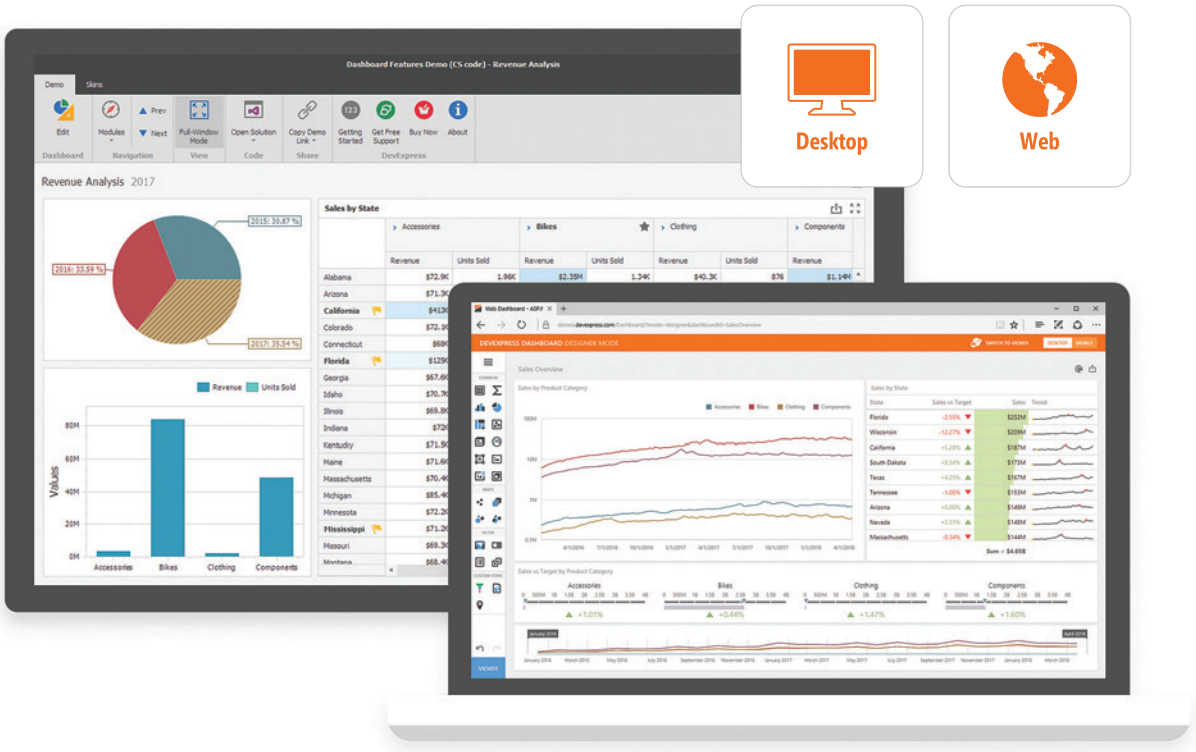
# msdn magazine

5

DevExpress®

# DevExpress Dashboard for .NET

Create and distribute royalty-free decision support systems and effortlessly share business intelligence across your entire enterprise.




Data Grid · Pivot Grid · Card View · Charts · Gauges · Treemap · Map · Range Control · Filter Control · Rich Text Editor · Image

With DevExpress Dashboard for .NET, creating insightful and information-rich decision support systems for executives and business users across platforms and devices is a simple matter of selecting the appropriate UI element (Chart, Pivot Table, Data Card, Gauge, TreeMap, Map, Grid or simple Filter elements) and dropping data fields onto corresponding arguments, values, and series. And because DevExpress Dashboard automatically provides the best data visualization option for you, results are immediate, accurate and always relevant.

## Learn More Today — Try DevExpress Dashboard Risk Free for 30 days

**devexpress.com/dashboard**

THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2019 VOL 34 NO 7

# msdn magazine

5

The Road to .NET 5.....12



## COLUMNS



Microsoft

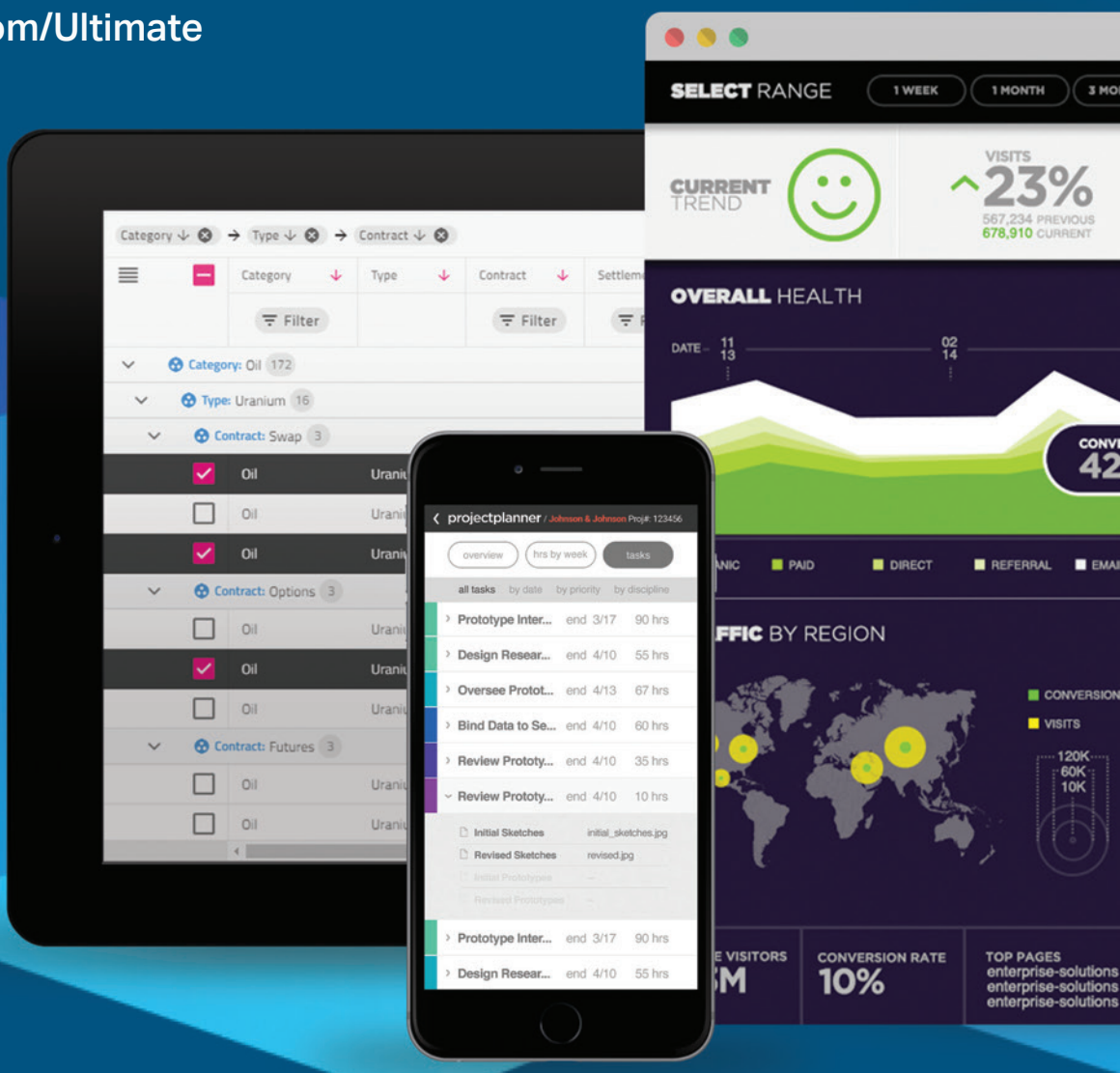
INFRAGISTICS

# 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 | React | 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 19.1

- Fastest **grids** & **charts** on the market – any device, any platform
- Build Spreadsheets with Charts in WPF, Windows Forms, Angular & JavaScript
- Get Angular code from Sketch designs with Indigo.Design
- 100% support for .NET Core 3

msdn magazine

JULY 2019 VOLUME 34 NUMBER 7



**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, File 2272, 1801 W.Olympic Blvd., Pasadena, CA 91199-2272, 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
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
Web: 1105Reprints.com

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

**Reaching the Staff**

Staff may be reached via e-mail, telephone, fax, or mail.

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

Irvine Office (weekdays, 9:00 a.m.-5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
2121 Alton Pkwy., Suite 240, 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

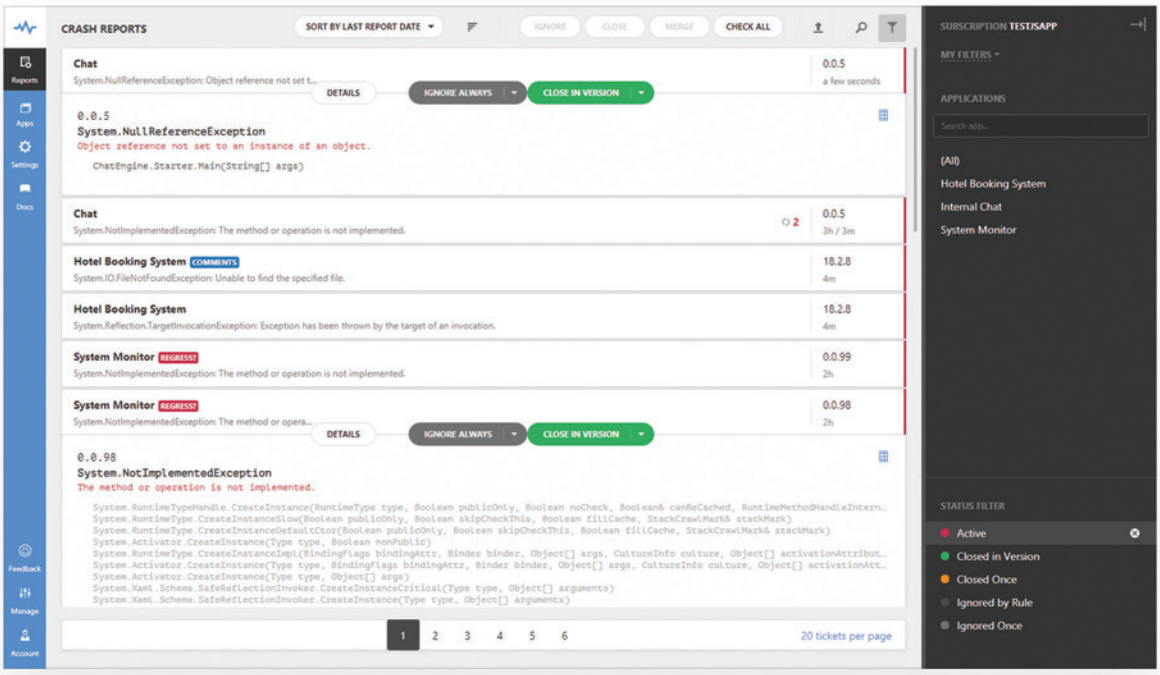
1906 2006 100 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

logify™

# Detect Errors and Exceptions
## with 24/7 Application Monitoring

Logify's cloud-based application monitoring service allows you to automatically collect app crash events and runtime exceptions. With Logify, you will never have to deal with Visual Studio's® internal exception information. Logify presents all relevant exception data in an easy-to-understand, clutter-free manner. From loaded modules and cookies to browser info, OS build, user activity — Logify will organize results so you can address application issues in the shortest possible time.




Learn More Today — Try Logify Risk Free for 15 days

**devexpress.com/logify**

DevExpress®

# Editor's Note

MICHAEL DESMOND

# Unification Theory

When Microsoft announced .NET 5 at the Build Conference in May, it drew a hearty round of applause from the developer audience, and for good reason. This future version of .NET Core (due in November 2020) will provide a single target for developers across desktop, Web, mobile, cloud and devices. As Mark Michaelis describes in his feature article this issue, ".NET Reunified: Microsoft's Plans for .NET 5," the effort reaches clear back to the start of the century, to .NET Framework 1.0 and the Shared Source Common Initiative, and to intervening frameworks like Silverlight, Windows Phone and currently .NET Core.

".NET 5 solidifies the future of .NET as a stalwart platform," Michaelis says. "Obviously, it eliminates all the complication of multiple .NETs, which is not trivial for new developers who have to work between them. .NET 5 wraps that all up and brings .NET back into one product."

It'll be a welcome reunion. Today, there are three primary managed frameworks in Microsoft's fleet: The flagship .NET Framework, the cloud and cross-platform friendly .NET Core that started as a subset of .NET Framework, and the device-savvy Mono/Xamarin framework for Android and iOS app development. There's also the .NET Standard reference API, which provides a tangible target for developers writing code that can run across .NET frameworks.

As Michaelis describes, the burden of maintaining and orchestrating three discreet frameworks escalated, even as they increasingly shared functionality. With .NET 5, Microsoft is focusing its vision squarely on .NET Core, which is currently in preview builds for version 3.0. As Microsoft Senior Program Manager Richard Lander told Michaelis, the plan sets the stage for innovation to come.

"Each of the .NET Core releases have been exciting because they've added new scenarios that have enabled larger groups of developers to adopt the platform and all the value that is part of it," Lander says. ".NET 5 is a major inflection point where all existing modern .NET scenarios provided by Microsoft will be supported with .NET Core. From there, we can focus on new scenarios like Web assembly and improving fundamentals like performance."

As Michaelis describes, the burden of maintaining and orchestrating three discreet frameworks escalated, even as they increasingly shared functionality.

Michaelis says the community has welcomed the plan to collapse development to a single, unified framework. Still, challenges remain. Microsoft says .NET 5 won't support APIs for ASP.NET Web Forms, Windows Workflow and Windows Communication Foundation (WCF) Server, which may force workarounds for developers engaged with those platforms.

"If you have a significant WCF Server implementation, there's nothing forcing you to migrate," Michaelis says. "You could keep it as is and then write all new code in .NET Standard and .NET Core assemblies that are referenced from the WCF Server code base. That way new stuff goes into the new .NET, while keeping the old stuff running."

What do you think of Microsoft's effort to unify development? Send me your thoughts at mdesmond@1105media.com.


## DevExpress DXperience 19.1 | from $1,439.99

DevExpress

**A comprehensive suite of .NET controls and UI libraries for all major Microsoft dev platforms.**

- WinForms – New Sunburst Chart, Office Navigation UX, SVG Office 2019 skins
- WPF – New Gantt control, improved Data Filtering UX and App Theme Designer
- ASP.NET & MVC – New Adaptive Layouts, improved Rich Text Editor and Spreadsheet
- Reporting – Vertical Band support, Free-hand drawing and improved Report wizards
- JavaScript – New HTML/Markdown WYSIWYG editor, Improved Grid and TreeList performance


## PBRS (Power BI Reports Scheduler) | from $9,811.51

Christiansteven

**Date & time Scheduling for Power BI reports with one Power BI License.**

- Exports reports to PDF, Excel, Excel Data, Word, PowerPoint, CSV, JPG, HTML, PNG and ePub
- Send reports to email, printer, Slack, Google Sheets, folder, FTP, DropBox & SharePoint
- Uses database queries to automatically populate report filters, email addresses & body text
- Adds flexibility with custom calendars e.g. 4-4-5, holidays, "nth" day of the month, etc.
- Responds instantly by firing off reports when an event occurs e.g. database record is updated


## LEADTOOLS Medical Imaging SDKs V20 | from $4,995.00 SRP

LEADTOOLS
THE WORLD LEADER IN IMAGING SDKs

**Powerful DICOM, PACS, and HL7 functionality.**

- Load, save, edit, annotate & display DICOM Data Sets with support for the latest specifications
- High-level PACS Client and Server components and frameworks
- OEM-ready HTML5 Zero-footprint Viewer with 3D rendering support and DICOM Storage Server
- Medical-specific image processing functions for enhancing 16-bit grayscale images
- Native libraries for .NET, C/C++, HTML5, JavaScript, WinRT, iOS, OS X, Android, Linux, & more


## Intellifront BI | from $23,246.35

Christiansteven

**Cost-Effective Data Analytics & Business Intelligence.**

- Design and serve visually stunning real-time reports using Grids, Charts, Gauges, Pies & more
- Add automatically updating KPI cards with aggregation, rounding, units, goals, colors & glyphs
- Create & assign Reporting, KPIs and Dashboards as consolidated "Canvases" to user groups
- Give your users a simple and intuitive self-service portal for their BI reports and documents
- Secure User Access with Active Directory Integration, Single Sign On & 2-Factor Authentication



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

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R



TED NEWARD

# Coding Naked: Naked Networking

Welcome back, NOFers. Last time, I delved into NakedObjects actions, which collect the business logic for a given business object into a method that the NOF framework makes available to the user for direct manipulation and use. To a certain degree, I want to continue that plunge, because in the stock NOF Angular client, the invocation of an action in the client somehow ends up on the server, from whence it can write changes to the database. Somehow, somewhere, a network traversal is made to get from client to server, and it's important this be explored, lest it get buried beneath hype.

At the risk of one more clothing-optional pun, are you ready to network naked?

## Really RESTing

To understand the NOF Restful API networking standard/protocol, you first need to grasp some key concepts. I want to discuss REST, but not the REST that's commonly tossed around in a cavalier way by architects and marketing hype machines. I need to go back to its origins and examine the source material.

Back in 2000, when Roy Fielding published the doctoral dissertation that served as the birthplace of REST, a new networking movement was born, although, to be fair, it would be more accurate to say that an existing networking approach was dissected and described. Fielding's dissertation is a masterful study of distributed systems design, and is highly recommended reading for anyone who wants to understand how to build distributed systems more effectively. The part that most people concern themselves with is in Chapter 5 of the document, entitled "Representational State Transfer (REST)." (The entire dissertation is available at bit.ly/1NbHM8Y or, if you prefer, you can jump directly to Chapter 5 at bit.ly/1eTY8Al.)

Readers new to Fielding's dissertation will find Chapter 5 especially interesting (although, honestly, the entire document is interesting and definitely worth reading), because Fielding deliberately "builds up" the REST architectural style (and he is quite deliberate about terminology here) from scratch, starting with what he calls the *null style*: "the null style describes a system in which there are no distinguished boundaries between components." He slowly works through a series of other styles, noting how in each case, there's an element of the style that's useful for building out the World Wide Web—which is the case study for REST.

The key area of the document to which you should turn your attention is in section 5.1.5, "Uniform Interface." I quote the paragraph, in its entirety, here:

> *The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components. By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. Implementations are decoupled from the services they provide, which encourages independent evolvability. The trade-off, though, is that a uniform interface degrades efficiency, since information is transferred in a standardized form rather than one which is specific to an application's needs. The REST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.*

There's a pretty hefty amount of concept to be unpacked here, but it's the last sentence in particular that provides the key to understanding the NOF Restful API: "hypermedia as the engine of application state" means that the entire state of the network interaction is stored in a hypermedia document, shared between the client and the server. In traditional Web scenarios, that hypermedia document is HTML, sent back to the client, with no further traces of client identity left on the server, which in turn allows the server to be entirely stateless with respect to the client. (It's for this reason of statelessness, by the way, that Fielding chooses to ignore the concept of cookies-as-client-identifiers on the server; in fact, both Fielding—and the HTTP standards—ignore cookies entirely.)

What this means, in practice, is that if a network interaction is going to characterize itself as "RESTful," it needs to obey this principle of "uniform interface" and capture all state in hypermedia—that is to say, in a document format that contains the "next steps" allowed by the client, a document that's exchanged between client and server.

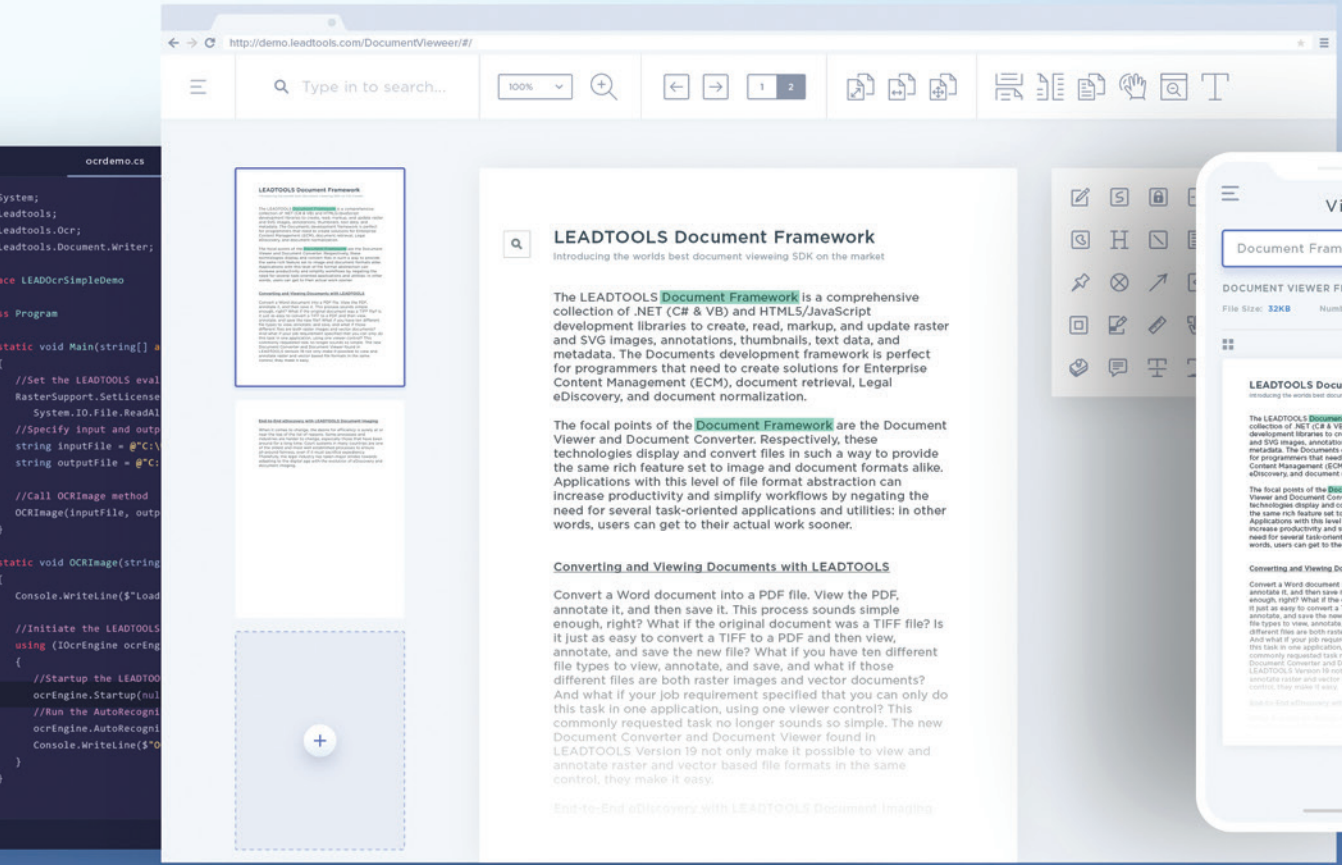
(By the way, take note that Fielding has made it clear that REST is not always superior: "The REST interface is designed to be efficient for large-grain hypermedia data transfer, … but [results] in an interface that is not optimal for other forms of architectural interaction." You will not hurt Fielding's feelings if you use something other than REST for your next project.)

## Restful API

The NOF Restful API, described in Fielding's dissertation, describes an HTTP-based protocol that looks to embrace REST principles with all their benefits and drawbacks. Specifically, the Restful API constrains all of its communication around the hypermedia-as-the-engine-of-application-state (dubbed HATEOAS

# LEADTOOLS®

## Multi-platform Document Viewer SDK




**With only a few lines of code, developers can use the LEADTOOLS Document Viewer SDK to add rich document viewing features to any project, including text search, annotation, memory-efficient paging, inertial scrolling, and vector display.**

- VIEW, CREATE, ORGANIZE & EXPORT A DOCUMENT FROM MULTIPLE FILES
- LOAD, SAVE, CONVERT PDF, DOC, DOCX, RTF, HTML, SVG, AND XPS
- VIEW AND EDIT ANNOTATIONS & MARKUP, INCLUDING PDF AND IBM FILENET
- DRAG AND DROP INTERFACE FOR ADDING AND REARRANGING PAGES
- INTELLIGENT TEXT EXTRACTION USES OCR ONLY WHEN NEEDED

Microsoft .NET · macOS · iOS

**Get Started Today**

DOWNLOAD OUR FREE EVALUATION

**LEADTOOLS.COM**

by REST practitioners and voted "Worst Acronym Ever" by the author of my personal blog), so that each exchange from a NakedObjects client is done in a full hypermedia request, and results in a full hypermedia document result. The easiest way to see this, of course, is to see it in action, so if the NOF Conference project isn't already running on your machine, fire it up. This time, however, instead of pointing the browser to localhost:5001, where the Angular project will be downloaded to your browser, point it at localhost:5000 instead, which is the port on which the server is listening for Restful API requests from the browser.

The initial request, http://localhost:5000, yields the JSON-based result shown in **Figure 1**.

That's a fat stream of gibberish, at least until it gets unpacked. But consider the structure that's already apparent: It's a JSON document, consisting of a root object that has two fields, "links" and "extensions." The latter is empty, but the former is an array of "rel"/"method"/"type"/"href" fields, in this case five of them, all GET requests, with the "href" field offering up … well, exactly what it sounds like, a hyperlink reference.

Let's try one of the simpler hyperlinks, the last one listed, which has "version" in the link. Popping that into the browser bar yields the results shown in **Figure 2**.

Sure enough, it's more of the same: This time the top-level object has "specVersion" and "implVersion" for the versions of the Restful Objects Specification, plus the "links" array you saw earlier (along with a few other fields that aren't relevant to the discussion at hand).

If, on the other hand, you navigate to the "user" link mentioned in the first request (http://localhost:5000/user), you get what's shown in **Figure 3**.

This provides the now-familiar "links" array, along with "userName" and "roles."

To get to any objects of interest, such as my speakers, in the Angular application, I can eye-scan the menu that's presented off of the homepage. To the Restful API, this means I need to examine the "menu" link from the homepage to see the options. Within that returned JSON (which I'll omit here to avoid wasting more column inches on unformatted JSON output), I get a "value" array that consists of several objects, each of which have a "title" (the menu item text), such as "Speakers" and "Talks," and a corresponding "href," which gives me links to the SpeakerRepository and TalkRepository, respectively. Not surprisingly, these links in turn provide links to the actions offered up by those services (each within its own submenu), and so on.

Figure 1 **The Initial Hypermedia Request**

```
{
  "links":[
  {
    "rel":"self",
    "method":"GET",
    "type":"application/json;
      profile=\"urn:org.restfulobjects:repr-types/homepage\"; charset=utf-8",
    "href":"http://localhost:5000/"
  },
  {
    "rel":"urn:org.restfulobjects:rels/user",
    "method":"GET",
    "type":"application/json; profile=\"urn:org.restfulobjects:repr-types/user\";
      charset=utf-8",
    "href":"http://localhost:5000/user"
  },
  {
    "rel":"urn:org.restfulobjects:rels/services",
    "method":"GET",
    "type":"application/json; profile=\"urn:org.restfulobjects:repr-types/list\";
      charset=utf-8; x-ro-element-type=\"System.Object\"",
    "href":"http://localhost:5000/services"
  },
  {
    "rel":"urn:org.restfulobjects:rels/menus",
    "method":"GET",
    "type":"application/json; profile=\"urn:org.restfulobjects:repr-types/list\";
      charset=utf-8; x-ro-element-type=\"System.Object\"",
    "href":http://localhost:5000/menus
  },
  {
    "rel":"urn:org.restfulobjects:rels/version",
    "method":"GET",
    "type":"application/json; profile=\"urn:org.restfulobjects:repr-types/version\";
      charset=utf-8",
    "href":http://localhost:5000/version
  }],
  "extensions":{}
}
```

Figure 2 **A Simple Hyperlink**

```
{
  "specVersion":"1.1",
  "implVersion":"8.1.1\r\n",
  "links":[
    {
      "rel":"self",
      "method":"GET",
      "type":"application/json;
        profile=\"urn:org.restfulobjects:repr-types/version\";
        charset=utf-8",
      "href":http://localhost:5000/version
    },
    {
      "rel":"up",
      "method":"GET",
      "type":"application/json;
        profile=\"urn:org.restfulobjects:repr-types/homepage\";
        charset=utf-8",
      "href":http://localhost:5000/
    }
  ],
  "extensions":{},
  "optionalCapabilities":{
    "protoPersistentObjects":"yes",
    "deleteObjects":"no",
    "validateOnly":"yes",
    "domainModel":"simple",
    "blobsClobs":"attachments",
    "inlinedMemberRepresentations":"yes"
  }
}
```

Figure 3 **Results from the User Link**

```
{
  "links":[
    {
      "rel":"self",
      "method":"GET",
      "type":"application/json; profile=\"urn:org.restfulobjects:repr-types/user\";
        charset=utf-8",
      "href":"http://localhost:5000/user"
    },
    {
      "rel":"up",
      "method":"GET",
      "type":"application/json;
        profile=\"urn:org.restfulobjects:repr-types/homepage\";
        charset=utf-8",
      "href":"http://localhost:5000/"
    }
  ],
  "extensions":{},
  "userName":"",
  "roles":[]
}
```

# 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

In short, the Restful API provides exactly what Fielding originally described in the REST chapter: hypermedia, in this case in the form of JSON documents containing embedded links, as the engine of the application state. This is as close to REST as you get, without building your own browser. (Which, technically, when you think about it, is exactly what a Naked Objects client is: a browser built for interaction with domain objects instead of HTML.)

The Restful API isn't the first time this kind of interaction has been discussed, by the way; the OData specification pioneered by Microsoft was another such effort that (unfortunately) fell on the rocks among the non-Microsoft platforms in the world. But before that, numerous *digiterati* spoke of how to accomplish this same kind of interaction using JSON or RSS XML.

One of the clear benefits of this approach is in testing: To test a particular endpoint, if the entire service is stateless, you need only submit the properly attributed and parameterized request and the server should respond with the correct response. In some cases, particularly for those services that must remain behind an authorization wall, this may require an extra request/response cycle, but it certainly still comes out ahead compared to the deeply nested series of calls that an RPC-style HTTP API normally commands.

## Wrapping Up

There are a number of interesting concepts still remaining in the Naked Objects Framework, but that's true of just about any topic this column ever covers, and it's time to move onward. Bear in mind that the point of this series is not a blatant push for you to use NOF in your own projects—although I heartily endorse the idea if the circumstances favor it—but for you to also consider how you might use some of these concepts in your own custom projects. Naked objects, as you've already seen, are the ultimate expression of the domain object concept, and if your project has reached the state of ubiquitous language and can clearly identify the domain objects, but expects a certain amount of churn and change over time as the business grows and evolves, it might be useful to build a mini-NOF framework to display and manipulate those objects. Similarly, it's all but impossible to have an inconsistent user interface when the UI is generated on-the-fly by examining the objects at runtime. And let's not get started on versioning—having a single client that knows how to display and manipulate objects based on their runtime characteristics is strong proof against needing to roll out a new client version every time a domain object's properties or actions change in response to business requirements.

Love them or hate them, Naked Objects is a powerful tool to have in your belt, and something every developer should spend some time exploring before moving on.

Speaking of which, it's high time we were moving on, as well. For now … happy coding! ■

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

**Thanks** to the following technical expert for reviewing this article: *Richard Pawson*

# .NET Reunified: Microsoft's Plans for .NET 5

Mark Michaelis

When Microsoft announced .NET 5 at Microsoft Build 2019 in May, it marked an important step forward for developers working across desktop, Web, mobile, cloud and device platforms. In fact, .NET 5 is that rare platform update that unifies divergent frameworks, reduces code complexity and significantly advances cross-platform reach.

This is no small task. Microsoft is proposing to merge the source code streams of several key frameworks—.NET Framework, .NET Core and Xamarin/Mono. The effort will even unify threads that separated at inception at the turn of the century, and provide developers one target framework for their work.

The source code flow concept in **Figure 1** shows how the timeline for each framework syncs up, and ultimately merges into a single thread as .NET 5 in November 2020. (Note that .NET Framework has been shortened to .NET FW in the image.) When released, .NET 5 will eclipse .NET Framework 4.8, Mono 5.0, and .NET Core 3.0.

Admittedly, **Figure 1** is more conceptual than reality, with source code forking (more likely just copying) rather than branching, and

This article discusses:

- The features and capabilities of the .NET 5 framework
- The evolution of .NET Core, Xamarin and .NET Framework toward a unified target
- Considerations for developers adapting to the new unified scheme

Technologies discussed:

.NET 5, .NET Framework, .NET Core, Xamarin/Mono

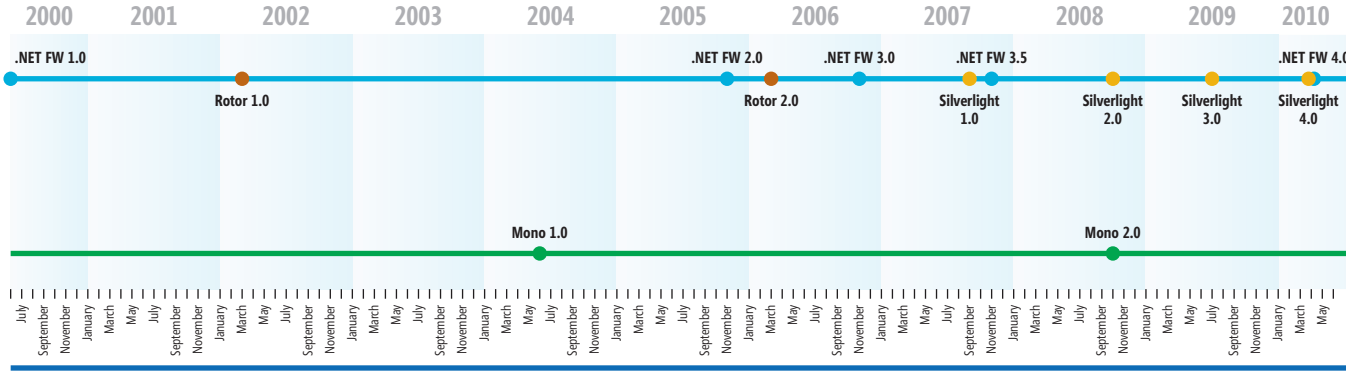



Figure 1 **Source Code Flow Concept from .NET, Mono, and Shared Source Initiative to .NET 5**

features (like Windows Presentation Foundation [WPF] or Windows Forms) migrating rather than merging. Still, the infographic provides a reasonably transparent view of the ancestral history of the .NET source code, showing its evolution from the three major branches all the way to .NET 5.

The result of this work is a unified platform with the .NET 5 framework executing on all platforms (desktop, Web, cloud, mobile and so on). **Figure 2** depicts this unified architecture.

## Origin Story

It's fascinating how the different .NET frameworks—such as Microsoft's Shared Source Common Initiative (Rotor), SilverLight, Windows Phone, .NET Core, and .NET Framework (but not Mono)—were originally compiled from the same source code. In other words, all the source code was maintained in a single repository and shared by all .NET frameworks. In so doing, Microsoft was able to ensure that APIs that were common to different frameworks would come from the same source code and have the same signatures. The only differences were which APIs were shared. (Note the use of .NET "framework," lowercase, referring to all .NET frameworks in general, vs .NET "Framework," uppercase, referring to the Windows .NET Framework, such as .NET Framework 4.8.)

To achieve a single source targeting different frameworks, various subsetting techniques, such as a multitude of #ifdefs, were used. Also of interest is how an orthogonal set of subsetting techniques (that is, different #ifdefs) were baked into the .NET source code to build Rotor's cross-platform support, enabling rapid development of both Silverlight and (much later) .NET Core from the same code base.

While the orthogonal set of subsetting techniques remains, the one that enables cross-platform compilation—the subsetting to produce the different frameworks—is being removed. (See, for example, the pull request at bit.ly/2WdSzv2 that removes several #ifdefs that are stale.) The reason it's possible to remove them today is because within days after the .NET 1.1 release, the .NET Core and .NET Framework source code was forked (more like copied). This in fact explains why there are two separate .NET source code Web sites: .NET Framework at referencesource.microsoft.com and .NET Core at source.dot.net. These are two separate code bases.

The decision to fork rather than continue the use of subsetting reflects the tension between maintaining backward compatibility (a high priority for the .NET Framework) and innovation (the priority of .NET Core). There were simply too many cases where maintaining compatibility conflicted with correcting or improving the .NET APIs, such that the source code had to be separated if both goals were to be achieved. Using #ifdefs was no longer a functional way to separate out the frameworks when, in fact, the APIs were different and version incompatible.

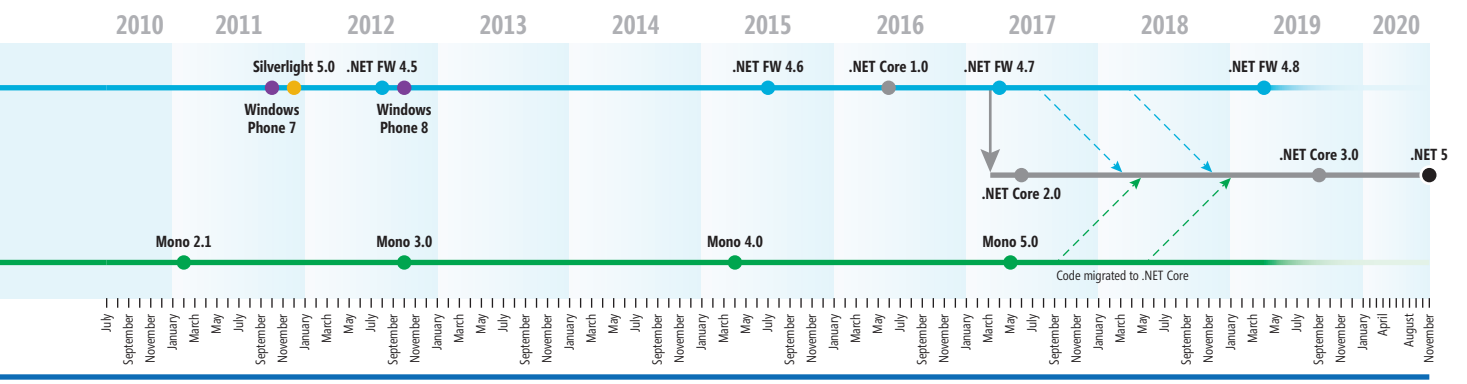
Over time, however, another conflict arose—that of allowing developers to create libraries that could successfully execute within both frameworks. To achieve this, an API standard was needed to assure developers that a framework would have a specific set of APIs identified by the standard. That way, if they leveraged only the APIs within the standard, their library would be cross-framework compatible (the exact same assembly could run on different frameworks—without even recompiling).

With each new feature added to .NET (for example Span<T>), it became more and more difficult to maintain backward compatibility with older versions of the framework. More specifically, the challenge was to support concepts in new .NET Standard versions—which were really new .NET Core innovations—in the .NET Framework. Furthermore, though less significantly, each new version of the .NET Standard included a larger and larger set of APIs until it became a maintenance burden to preserve .NET Standard compatibility between .NET Framework and .NET Core.

## Come Together

The two frameworks began to look more and more alike because of the standard. As the APIs became more consistent, the obvious question began to arise: Why not move the separate code bases back together? And, in fact, starting with .NET Core 3.0 preview, so much of the .NET Framework WPF and Windows API was cherry-picked and merged into the .NET Core 3.0 code base, that this is exactly what happened. The source code for .NET Core 3.0 became one and the same with the modern day (desktop, cloud, mobile and IoT) functionality in .NET Framework 4.8.

At this point there's still one major .NET framework that I haven't covered: Mono/Xamarin. Although the source code for Rotor was publicly available, using it would've violated the license agreement. Instead, Mono began as a separate green field development effort with the vision to create a Linux-compatible version of .NET. The Mono framework continued to grow over time, until the company (Ximian) was acquired in 2003 by Novell and then shuttered eight years later following Novell's sale to Attachmate. Ximian manage-



ment quickly reformed in May 2011 as Xamarin. And less than two years later, Xamarin developed a cross-platform UI code base that ran on both Android and iOS, leveraging a now closed-source, cross-platform version of Mono under the covers.

In 2016 Microsoft acquired Xamarin to bring all the .NET framework source code under the control of a single company. Shortly after, Mono and the Xamarin SDK would be released as open source.

This brings us to where we are in the first half of 2019, with essentially two main code bases going forward: .NET Core 3.0 and Mono/Xamarain. (While Microsoft will support .NET Framework 4.8 on Windows for as long as anyone can forecast, .NET Core 3.0 and later .NET 5 will eclipse it as the strategic platform for new applications going forward.) Alongside this is the .NET Standard and the unification of APIs into the soon-to-be-released .NET Standard 2.1.

Again, the question arises, with APIs moving closer and closer, can we not merge .NET Core 3.0 with Mono? It's an effort, in fact, that has already begun. Mono today is already one-third Mono source, one-third CoreFx, and one-third .NET Framework reference source. That set the stage for .NET 5 to be announced at Microsoft Build 2019.

## The Advantages of .NET 5

This unified version of .NET 5 will support all .NET application types: Xamarin, ASP.NET, IoT and desktop. Furthermore, it will leverage a single CoreFX/Base Class Library (BCL), two separate runtimes and runtime code bases (because it's really hard to single source two runtimes intended to be critically different), and a single tool chain (such as dotnet CLI). The result will be uniformity across behaviors, APIs and developer experiences. For example, rather than having three implementations of the System.* APIs, there will be a single set of libraries that run on each of the different platforms.

There are a host of advantages with the unification of .NET. Unifying the framework, runtimes, and developer toolsets into a single code base will result in a reduction in the amount of duplicate code that developers (both Microsoft and the community) will need to maintain and expand. Also, as we've come to expect from Microsoft these days, all the .NET 5 source code will be open source.

With the merger, many of the features exclusive to each individual framework will become available to all platforms. For example, csproj types for these platforms will be unified into the well-loved and simple .NET Core csproj file format. A .NET Framework project type, therefore, will be able to take advantage of the .NET Core csproj file format. While a conversion to .NET Core csproj file formats is necessary for Xamarin and .NET Framework (including WPF and Windows Forms) csproj files, the task is similar to the conversion from ASP.NET to ASP.NET Core. Fortunately, today it's even easier to do thanks to tools like ConvertProjectToNETCore3 (see bit.ly/2W5Lk3D).

Another area of significant difference is in the runtime behavior of Xamarin and .NET Core/.NET Framework. The former uses a static compilation model, with ahead-of-time (AOT) compilation that compiles source code down to the native source code of the platform. By contrast, .NET Core and .NET Framework use just-in-time (JIT) compilation. Fortunately, with .NET 5, both models will be supported, depending on the project type target.

For example, you can choose to compile your .NET 5 project into a single executable that will use the JIT compiler (jitter) at runtime, or a native compiler to work on iOS or Android platforms. Most projects will leverage the jitter, but for iOS all the code is AOT. For client-side Blazor, the runtime is Web Assembly (WASM), and Microsoft intends to AOT compile a small amount of managed code (around 100kb to 300kb), while the rest will be interpreted. (AOT code is large, so the wire cost is quite a burden to pay.)

In .NET Core 3.0 you can compile to a single executable, but that executable is actually a compressed version of all the files needed to execute at runtime. When you execute the file, it first expands itself out into a temporary directory and then executes the entry point of the application from the directory that contains all the files. By contrast, .NET 5 will create a true, single-executable file that can execute directly in place.

Another remarkable feature of .NET 5 is interoperability with source code from Java and Objective-C (including Swift). This has been a feature of Xamarin since the early releases, but will extend to all .NET 5 projects. You'll be able to include jar files in your csproj file, for example, and you'll be able to call directly from your .NET code into Java or Objective-C code. (Unfortunately, support for Objective-C will likely come later than Java.) It should be noted that interoperability between .NET 5 and Java/Objective-C is only targeted at in-process communication. Distributed communication to other processes on the same machine or even processes on a different machine will likely require serialization into a REST- or RPC-based distributed invocation.

## What's Not in .NET 5

While there's a significant set of APIs available in the .NET 5 framework, it doesn't include everything that might have been developed over the last 20 or so years. It's reasonable to expect that all the APIs identified in .NET Standard 2.1 will be supported, but some of the

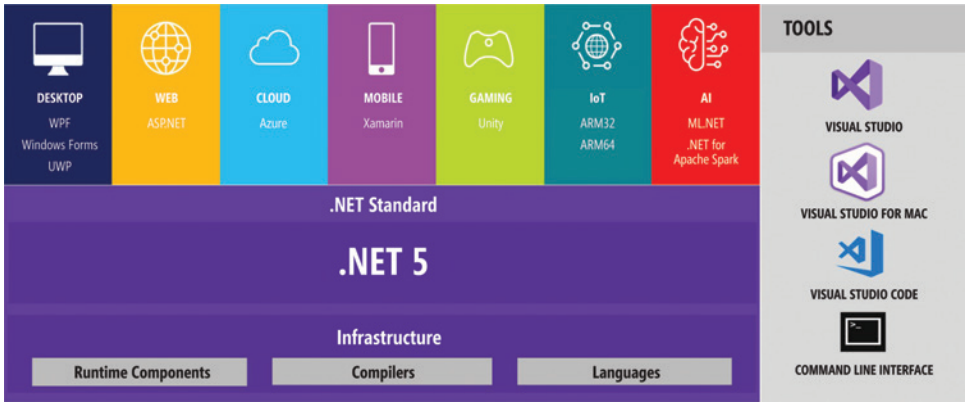
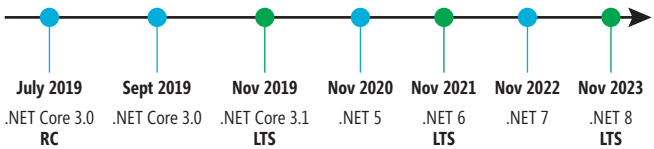



Figure 2 **.NET 5—a Unified Platform**




- .NET Core 3.0 release in September
- .NET Core 3.1 = Long Term Support (LTS)
- .NET 5 release in November 2020
- Major release every year, LTS for even-numbered releases
- Predictable schedule, minor releases if needed

Figure 3 .NET Release Schedule

more "legacy" APIs, including Web Forms, Windows Communication Foundation (WCF) server and Windows Workflow, will not. These are destined to remain in .NET Framework only. If you wish to achieve the same functionality within .NET 5, consider porting these APIs as follows:

- ASP.NET Web Forms => ASP.NET Blazor
- WCF server and remoting => gRPC
- Windows Workflow (WF) => Core WF (github.com/UiPath/corewf)

The lack of WCF server support is no doubt disappointing to some. However, Microsoft recently decided to release the software under an MIT open source license, where its destiny is in control of the community (see github.com/CoreWCF/CoreWCF). There's still a tremendous amount of work to be done to release independently of the .NET Framework, but in the meantime, the client-side WCF APIs are available (see github.com/dotnet/wcf).

## A Declaration of Intent

Even as Microsoft makes plans to unify its developer frameworks under .NET 5, the company has announced that it's adopting a regular cadence for its unified .NET releases (see **Figure 3**). Going forward, you can expect general availability versions of .NET to be released in Q4 of each year. Of these releases, every second version will be a Long Term Support (LTS) release, which Microsoft will support for a minimum of three years or one year after a subsequent LTS release, whichever is longer. In other words, you'll always have at least three years to upgrade your application to the next LTS release. See bit.ly/2KfkkW0 for more information on the .NET Core support policy, and what can be reasonably expected to become the .NET 5 and beyond support policy.

At this point, .NET 5 is still just an announcement—a declaration of intent, if you like. There's lots of work to be done. Even so, the announcement is remarkable. When .NET Core was first released, the goal was to provide a cross-platform .NET version that could highlight Azure (perhaps especially the Platform-as-a-Service [PaaS] portions of Azure and support for .NET on Linux and within Linux containers).

In the original concept, the idea that all of .NET Framework could be ported to .NET Core wasn't considered realistic. Around the time of the .NET Core 2.0 release, that began to change. Microsoft realized that it needed to define a framework standard for all .NET framework versions, to enable code running on one framework to be portable to another.

This standard, of course, became known as the .NET Standard. Its purpose was to identify the API that a framework needed to support so that libraries targeting the standard could count on a specific set of APIs being available. As it turned out, defining the standard and then implementing it with Xamarin/Mono, .NET Core, and .NET Framework became a key component that made the .NET 5 unification strategy possible.

For example, once each framework has implemented code that supports the .NET Standard set of APIs, it seems logical to work toward combining the separate code bases into one (a refactoring of sorts). And, where behavior isn't the same (JIT versus AOT compilation, for example), why not merge the code so that all platforms support both approaches and features? The effort isn't trivial, but the result is a huge step forward in reducing complexity and maintenance, while at the same time unifying the features to all platforms.

Perhaps surprisingly, the very .NET Standard that made unification possible will likely make .NET Standard irrelevant. In fact, with the emergence of .NET 5, it's doubtful there will be another version of .NET Standard—.NET 5 and each version after that *will* be the standard.

## Wrapping Up

They say timing is everything, and that's true of .NET 5. A virtually comprehensive re-write of the .NET Framework wasn't even conceivable when .NET Core development started. At the time, Microsoft was responding to demand to significantly enhance the Azure hosting experience on Linux, in containers, and on PaaS. As such, the company was laser-focused on getting something out to meet the demands of customers and the Azure product team.

With .NET Core 2.0 the mission expanded to matching the functionality found in the .NET Framework. Again, the team was laser-focused on releasing something viable, rather than taking on too much. But things began to change with .NET Core 3.0 and the implementation of .NET Standard 2.1. The idea of having to go in and make changes to three distinct frameworks when a new feature or bug came up was an irritation and an expense. And, like any good developer, the idea soon emerged to refactor the code as much as possible into a single code base.

And so, .NET 5 was born. And along with it was born the idea of unifying all the features of each framework—whether it was simple csproj formats, adopting open source development models, enabling interoperability with Java and Objective-C (including Swift), or supporting JIT and AOT compilation. Just like that, the idea of a single, unified framework became an obvious next step, and one I expect everyone both inside and outside of Microsoft will celebrate. ■

**Mark Michaelis** *is founder of IntelliTect, where he serves as its chief technical architect and trainer. For nearly two decades he has been a Microsoft MVP, and a Microsoft Regional Director since 2007. Michaelis serves on several Microsoft software design review teams, including C#, Microsoft Azure, SharePoint and Visual Studio ALM. He speaks at developer conferences and has written numerous books including his most recent, "Essential C# 7.0 (6th Edition)" (itl.tc/EssentialCSharp). Contact him on Facebook at facebook.com/Mark.Michaelis, on his blog at IntelliTect.com/Mark, on Twitter: @markmichaelis or via e-mail at mark@IntelliTect.com.*

**Thanks** to the follow Microsoft technical expert for reviewing this article: *Rich Lander*


# 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! Chicago, including everything Tuesday – Thursday at the conference.

**SAVE $400! When You Register by August 2**

Your Adventure Starts Here!

EVENT PARTNER Microsoft

GOLD SPONSOR CSC

SUPPORTED BY Visual Studio, msdn magazine, Visual Studio MAGAZINE

PRODUCED BY CONVERGE 360, An 1105 MEDIA company

vslive.com/chicago

# AGENDA AT-A-GLANCE

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

| START TIME | END TIME | | | | |
|---|---|---|---|---|---|
| | | **Pre-Conference Full Day Hands-On Labs: Sunday, October 6, 2019** *(Separate entry fee required)* | | | |
| 7:30 AM | 9:00 AM | Pre-Conference Workshop Registration - Coffee and Morning Pastries | | | |
| 9:00 AM | 6:00 PM | **HOL01** Full Day Hands-On Lab: Develop an ASP.NET Core 2 App with EF Core 2 in a Day - *Philip Japikse* | | **HOL02** Full Day Hands-On Lab: Azure and Xamarin: Build the Mobile Apps of Tomorrow with Serverless, AI and Cross-platform clients - *Laurent Bugnion & Brandon Minnick* | |
| **START TIME** | **END TIME** | **Post-Conference Workshops: Monday, October 7, 2019** *(Separate entry fee required)* | | | |
| 7:30 AM | 8:00 AM | Post-Conference Workshop Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 5:00 PM | **M01** Workshop: Go Serverless in the Azure Cloud with Azure DevOps - *Brian Randell* | **M02** Workshop: SQL Server for Developers: The Grand Expedition - *Andrew Brust and Leonard Lobel* | **M03** Workshop: Cross-Platform C# Using .NET Core, Kubernetes, and WebAssembly - *Rockford Lhotka & Jason Bock* | |
| 6:45 PM | 9:00 PM | Dine-A-Round | | | |
| **START TIME** | **END TIME** | **Day 1: Tuesday, October 8, 2019** | | | |
| 7:00 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **T01** Guiding Principles for ASP.NET Core - *K. Scott Allen* | **T02** Cross-Platform Development with Xamarin, C#, and CSLA .NET - *Rockford Lhotka* | **T03** AI and Analytics with Apache Spark on Azure Databricks - *Andrew Brust* | **T04** Signing Your Code the Easy Way - *Oren Novotny* |
| 9:30 AM | 10:45 AM | **T05** Angular 101 - *Tracy Lee* | **T06** Azure 101 - *Laurent Bugnion* | **T07** DevOps Practices Can Make You A Better Developer - *Robert Green* | **T08** A Tour of Visual Studio 2019 - *Jason Bock* |
| 11:00 AM | 12:00 PM | Keynote: To Be Announced | | | |
| 12:00 PM | 1:00 PM | Lunch | | | |
| 1:00 PM | 1:30 PM | Dessert Break - Visit Exhibitors | | | |
| 1:30 PM | 2:45 PM | **T09** Reactive Programming Using RXJS - RXJS Operators - Real World Use Cases, Anti Patterns & Debugging - *Tracy Lee* | **T10** To Be Announced | **T11** Power BI: What Have You Done For Me Lately? - *Andrew Brust* | **T12** What's New in .NET Core 3.0 - *Jason Bock* |
| 3:00 PM | 4:15 PM | **T13** Advanced Azure App Services - *K. Scott Allen* | **T14** (WPF + WinForms) * .NET Core = Modern Desktop - *Oren Novotny* | **T15** Get Started with Git - *Robert Green* | **T16** Modernize Your App to be Delivered as a SaaS Service - *Nick Pinheiro* |
| 4:15 PM | 5:30 PM | Welcome Reception | | | |
| **START TIME** | **END TIME** | **Day 2: Wednesday, October 9, 2019** | | | |
| 7:30 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **W01** Designing and Building Advanced Angular Components - *Anthony Monsees* | **W02** Azure Cosmos DB Part I - Introduction to Cosmos DB - *Leonard Lobel* | **W03** Reach Any User on Any Platform with Azure AD B2C - *Nick Pinheiro* | **W04** Azure Pipelines - *Brian Randell* |
| 9:30 AM | 10:45 AM | **W05** Advanced Typescript - Dive into Inheritance, Union Types, Generics, and More - *Anthony Monsees* | **W06** Azure Cosmos DB Part II - Building Cosmos DB Applications - *Leonard Lobel* | **W07** 9 Azure Services Every Developer Should Know About - *Eric D. Boyd* | **W08** Non-Useless Unit Testing Entity Framework & ASP.NET MVC - *Benjamin Day* |
| 11:00 AM | 12:00 PM | General Session: To Be Announced | | | |
| 12:00 PM | 1:00 PM | Birds-of-a-Feather Lunch | | | |
| 1:00 PM | 1:30 PM | Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win) | | | |
| 1:30 PM | 1:50 PM | **W09** Fast Focus: Getting Started with ASP.NET Core 2.0 Razor Pages - *Walt Ritscher* | **W10** Fast Focus: What's New in EF Core 2.x - *Jim Wooley* | **W11** Fast Focus: Serverless of Azure 101 - *Eric D. Boyd* | |
| 2:00 PM | 2:20 PM | **W12** Fast Focus: What is WSL and Why Do I Care? - *Brian Randell* | **W13** Fast Focus: A Real-time SignalR Core Chat - *Jim Wooley* | **W14** Fast Focus: Scrum in 20 Minutes - *Benjamin Day* | |
| 2:30 PM | 3:45 PM | **W15** Diving Deep Into ASP.NET Core 2.x - *Philip Japikse* | **W16** Azure Data Explorer— An In-depth Look at the New Microsoft PaaS offering - *Raj Krishnan* | **W17** Keep Secrets with Azure Key Vault - *Eric D. Boyd* | **W18** Getting to SAFe in the Enterprise - *Jim Szubryt* |
| 4:00 PM | 5:15 PM | **W19** JavaScript for the C# (and Java) Developer - *Philip Japikse* | **W20** Building End-to-End ML Solutions Using Azure Machine Learning Service - *Raj Krishnan* | **W21** Achieving DevSecOps with Azure DevOps and Containers - *Jim Szubryt* | **W22** Scrum Under a Waterfall - *Benjamin Day* |
| 7:00 PM | 9:00 PM | VSLive!'s Trolley Tour of the Windy City | | | |
| **START TIME** | **END TIME** | **Day 3: Thursday, October 10, 2019** | | | |
| 7:30 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **TH01** WebAssembly: the Browser is Your OS - *Jeremy Likness* | **TH02** UX Beyond the Keyboard: Designing Conversational Interfaces - *John Alexander* | **TH03** Building a Stronger Team, One Strength at a Time - *Angela Dugan* | **TH04** C# Language Enhancements, Roslyn and You - *Jim Wooley* |
| 9:30 AM | 10:45 AM | **TH05** What's New in Bootstrap 4 - *Paul Sheriff* | **TH06** How to Have Better Business Intelligence Through Visualizations - *Walt Ritscher* | **TH07** How Do You Measure Up? Collect the Right Metrics for the Right Reasons - *Angela Dugan* | **TH08** Get Your BASH on with WSL - *Brian Randell* |
| 11:00 AM | 12:15 PM | **TH09** Create Your Own SPA Using jQuery - *Paul Sheriff* | **TH10** Microsoft Power Platform, RAD Done Right: Building Dynamic Mobile Apps with PowerApps - *Walt Ritscher* | **TH11** Writing Testable Code and Resolving Dependencies - DI Kills Two Birds with One Stone - *Miguel Castro* | **TH12** Entity Framework Performance Monitoring and Tuning - *Jim Wooley* |
| 12:15 PM | 1:30 PM | Lunch | | | |
| 1:30 PM | 2:45 PM | **TH13** Real-time Web Applications with ASP.NET Core SignalR - *Roland Guijt* | **TH14** Building UWP Apps for Multiple Devices - *Tony Champion* | **TH15** Demystifying Microservice Architecture - *Miguel Castro* | **TH16** .NET Standard: Reuse All the Code - *Jonathan "J." Tower* |
| 3:00 PM | 4:15 PM | **TH17** Top 10 Browser Threats Mitigated - *Roland Guijt* | **TH18** Building Cross Device Experiences with Project Rome - *Tony Champion* | **TH19** Hacking Time: Using Micro-habits to Customize Your Environment - *John Alexander* | **TH20** Dotnet CLI or: How I Learned to Stop Worrying and Love the Command Line - *Jonathan "J." Tower* |

*Speakers and sessions subject to change*

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

# Understanding Azure AI Services

Ashish Sahu and Sagar Bhanudas Joshi

Cloud computing changed the way products and services are imagined and created. It changed how software solutions were architected and enabled developers to build apps on a planetary scale. And it paved the way to democratize artificial intelligence (AI) for developers across the globe.

Microsoft Azure leads the pack in empowering developers with a spectrum of AI services. But as so often happens when we're spoiled with choices, it becomes difficult to discern what services are the best ones to use and when. In this article, we walk through the AI services available on Microsoft Azure and provide the information and context you need to analyze your requirements and make informed decisions about which services to use based on your requirements. In some architectures, you might end up with a pipeline of multiple AI services, working in concert to achieve your goals.

Among the range of services available, Microsoft Azure provides more than 35 services targeted at AI and machine learning (ML).

This article discusses:

- Understanding the range of Azure AI tools and services
- Exploring Azure AI services in context
- Discovering the capabilities of Azure AI services

Technologies discussed:

Azure AI Platform, Microsoft Cognitive Services, Azure Machine Learning Services

These services range from ready-to-use RESTful APIs to the services and SDKs that you can use to create an AI model from scratch using your own data. As we go forward, we'll discuss these services and share code that you can try yourself. We will also explore the scenarios, requirements and target audience for these services, so you can understand when and what services make a case for themselves.

Among the range of services available, Microsoft Azure provides more than 35 services targeted at AI and ML.

As a practicing software or ML developer/architect, you may already be aware that the terms AI and ML refer to a vast domain of software tools, frameworks, algorithms and data processing techniques. The features that are powered by AI can range from the simple—say, a personalized recommendation system—to extremely complex—such as identifying actions in videos.

The AI and ML services on Azure cater to everything under the sun and include pre-built AI services that you can integrate into your solution with a few RESTful API calls. If your specific

requirement mandates a custom ML model, there are tools and services to build and use those, as well. Openness has been a core tenet of the Microsoft Azure platform, and the Microsoft AI platform allows you to use any toolset or framework that you use today, and still leverage Azure AI Services to streamline the ML workflow end-to-end.

## An Overview of AI Services

For simplicity, we can arrange AI Services into three high-level categories: AI Apps and Agents, Knowledge Mining and Machine Learning. Let's take a moment to explore these categories.

> For simplicity, we can arrange AI Services into three high-level categories: AI Apps and Agents, Knowledge Mining and Machine Learning.

**AI Apps and Agents:** As a category, AI Apps and Agents are a subset of services offered by Microsoft Cognitive Services and the Azure Bot Service. Microsoft Cognitive Services are pre-built AI services that allow developers to quickly add intelligent features in their apps and services with just a few RESTful API calls. Some of these services provide customization options to suit your specification requirements, but they're largely powered by models developed, trained and hosted by Microsoft, so you don't really need to go through the complete process of developing a model from scratch.

Azure Bot Service is powered by the Microsoft Bot Framework platform and provides you with ways to quickly get started with functioning bots that you can extend using Microsoft Cognitive Services or other AI services from Azure. You can also integrate any AI services that you're developing on your own. The Speech and Language Cognitive Services are also referred to as Conversational AI services in conjunction with Azure Bot Service.

**Knowledge Mining:** Knowledge Mining refers to a branch of Azure AI platform where intelligent information extraction is the key channel to surface insights from a corpus of structured and unstructured data. Microsoft Azure Search (with its Cognitive Skills) and Azure Form Recognizer service are two shining examples of services that are part of this category.

**Machine Learning:** Both the AI Apps and Agents and Knowledge Mining categories include pre-built AI services. However, Azure AI platform includes bespoke services that will make any data scientist feel at home. The Machine Learning category includes Microsoft AI Platform services to bring your ML models to life. These Azure services help you manage your ML experiments from data prep stages, to testing and managing the training, to the evaluation of runtimes. They also provide a flexible compute targets option so you can focus solely on your experiment code, rather than worrying about infrastructure, platforms and scaling. Azure Machine Learning Studio, Automated Machine Learning service (also with SDK), Azure Notebooks and Azure Machine Learning Services provide you with platform tools and services to make your experiment development more productive.

## Getting Practical

It's one thing to review the list of available services. It's quite another to understand how they work and interact in the context of a realistic scenario that applies to a broad range of organizations. With that in mind, we'll describe a scenario and then use that to show how specific Azure components can address associated tasks.

For our journey today let's start with a bit of background. Suppose you're a lead developer, architect or AI engineer at everyone's favorite company, Contoso. Contoso recently announced the modernization of its business processes, with one of the projects aimed to transform an expense management solution built in-house a decade ago. While the core business logic and approval workflows might not necessarily change, AI can be used to improve user interaction, data input and information processing.

To make this clearer, let's understand the possible tasks involved in the project and then assign a specific AI service to each task for implementation. **Figure 1** illustrates the tasks that are relevant to this project.

Now let's walk through the tasks in **Figure 1** and see which Microsoft AI service can be used for each task.

### Information Extraction from Receipts: Simple & Complex

In this scenario, the requirement is to automate information retrieval from scanned or digital receipts uploaded by users. This stage extracts fields, amounts, and vendor information from the

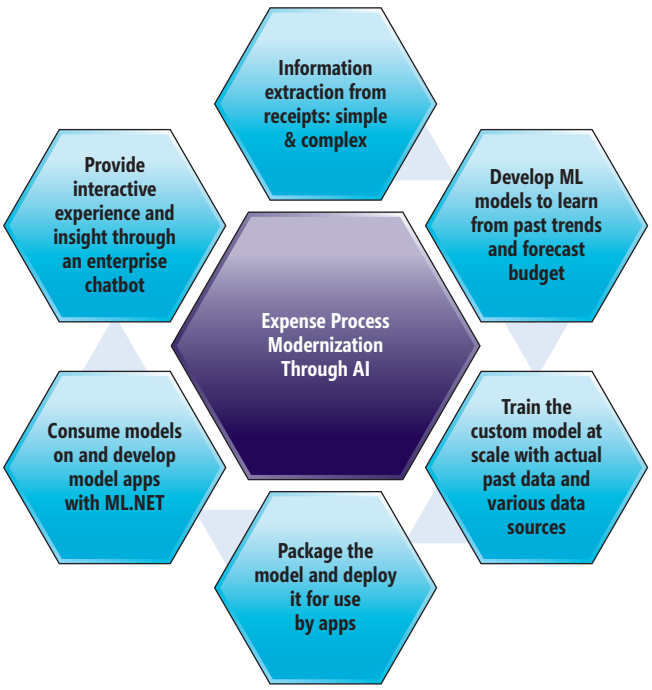



Figure 1 **Task Map for Contoso Expense Modernization**

receipts and pushes them to the data store or to a UI for review in the expense application.

**Relevant Services:** *Azure Cognitive Services, Azure Search, Azure Form Recognizer Service*

Scenarios that require scanning an image to recognize text (optical character recognition [OCR]) or a set of images and documents call for Azure Search with Cognitive Skills, Computer Vision, and Text Analytics Cognitive Services. The last service is used to extract entities and key phrases (for example, "Seattle" and "dinner") to complete this task.

**Target audience:** AI engineers and application developers.

**Hands-on Samples:**

- Quickstart: Create an AI indexing pipeline using cognitive skills in Azure Search (bit.ly/2WNDooN)
- Computer Vision API sample in JavaScript (bit.ly/2YCcFvB)
- Quickstart: Use the .NET SDK and C# to explore the Text Analytics service (bit.ly/2VKkyST)

## Develop ML Models to Learn from Past Trends and Forecast Budget

The data science community thrives on collaboration and knowledge sharing. But as the scale and complexity of both data and requirements increase, you may find reasons to build your own ML models, which are more effective in understanding the patterns in the data at large scale. These models are referred to as custom ML models.

The lifecycle of shipping intelligent features or building custom ML models is called Team Data Science Process (TDSP) or Data Science Lifecyle. The stages of TDSP bring structure to the development process just like an application development lifecycle. The process starts with an understanding of the business requirements and ends with customer acceptance. In between are intermediate stages—like data acquisition and understanding, and modeling and deployment--that can be iterated as needed. You can learn more about the data science lifecycle at bit.ly/2XbmKj3.

In our example, the in-house data science team is creating a statistical modeling of past spends and forecasting future expense patterns to help the finance department with planning next fiscal year. The data science experts sample data from an expense data store and perform Exploratory Data Analysis (EDA) using familiar tools. The resulting custom ML model is tuned to specific requirements and then learns from the gathered data.

**Relevant Services:** *Azure Data Science Virtual Machine (DSVM), Azure Notebooks, Azure ML Services, Azure ML Studio*

DSVM is a templated image of OS images built to speed up development of custom ML experiments, while Azure Notebooks and Visual Studio can provide a great environment for running data science experiment projects. Azure ML Services helps teams of data scientists to collaborate and maintain their models using Notebook VMs, code and compute environments, and manage the application lifecycle. Automated ML Service is also an option to generate high-performing models. You can also use Azure ML Studio if you'd rather experiment using a visual drag-and-drop interface. Azure Notebooks are another way to host Jupyter notebooks publicly or privately that can still connect to your ML compute for training and deployment.

**Target audience:** Data scientists and data scientist teams, developers, analysts, architects

**Hands-on Samples:**

- Azure Notebooks (notebooks.azure.com)
- Quickstart: Use a cloud-based notebook VM to get started with Azure ML (bit.ly/2WJEOR3)
- Tutorial: Use automated ML to build your regression model (bit.ly/2W7SbgH)
- Explore Azure ML Studio with Azure AI Gallery (gallery.azure.ai)

## Train the Custom Model at Scale with Actual Past Data and Various Data Sources

In this scenario, the data scientists in your organization want to run their models on actual data to understand the accuracy of their forecasts. One key factor here is to minimize the training time as much as possible.

**Relevant Services:** *Azure Databricks, Azure ML Services, Azure Data Lake*

Usually experimentation happens on sampled data to create custom ML models. After the model is developed with sample data, it's trained with actual data, which might be gigabytes or terabytes in size. Distributed computing and Big Data storage are essential for optimal performance when learning with data at this scale. Azure ML Services helps you to choose compute targets for training at scale, ranging from a GPU virtual machine (VM) to an Azure Kubernetes cluster. If there are Big Data experts on the team, you might choose to run the training-at-scale scenarios using Azure Databricks for the compute layer (for Spark/MMLSpark deployments) or using Azure Data Lake Analytics (if the data resides in Azure Data Lake).

Azure Databricks also provides a data-engineering platform to help with data ingestion, ETL and orchestration tasks.

**Target audience:** Data scientists, data engineers, data developers

**Hands-on Samples:**

- Quickstart: Run a Spark job on Azure Databricks using the Azure Portal (bit.ly/2EevltO)
- Tutorial: Extract, transform, and load data by using Azure Databricks - bit.ly/30pj3bs

## Package the Model and Deploy It for Use by Apps

The custom machine model is ready! While there are various ways to evaluate, most applications will interact with models through an API. Moreover, the hosting platform (OS) of your customer environment can vary based on a lot of factors. Hence, it's important for your DevOps teams and ML teams to work closely to ensure that models are accessible, available and can run cross-platform.

**Relevant Services:** *Azure IOT Edge, Azure Container Instances, Azure ML Services*

Container platforms have gained a lot of momentum in application hosting scenarios as they package apps and dependencies in one portable unit. Containers provide flexibility of model packaging in cloud, hybrid, edge or private hosting scenarios. Here you can use Azure ML Services to deploy the model files to a multitude of environments, such as Local Web Service, Azure IOT Edge, Azure Kubernetes Service, Azure Container Instances or Azure ML Compute. AI models exported from Computer Vision can be run as containers on various OS and orchestration platforms.

**Target audience:** Application developers, DevOps engineers, data scientists

**Hands-on Samples:**

- Install and run Computer Vision OCR in containers (bit.ly/2Jq6ozF)
- Deploy models with the Azure ML Services (bit.ly/2YwoHGY)

### Consume Models on and Develop Model Apps with ML.NET

Most of your Contoso line-of-business apps have been written in .NET Core, while some solutions run on the edge or as mobile apps. This is where the application teams and architects need to load the ML models developed by data science teams into the mobile and edge apps.

**Relevant Services:** *ML.NET, Azure Cognitive Services*

ML.NET is a .NET Core-based, cross-platform framework for developers familiar with .NET to start developing ML applications in their language of choice. You can load ONNX and TensorFlow models in your .NET Core apps through ML.NET. You can also create a model from data, save it and then load it for evaluation in other ML.NET apps. For full-stack application developers, there's an option to infuse AI into apps through a wide range of Azure Cognitive Services.

Models exported from some Cognitive Services can also be loaded into iOS and Android apps.

**Target audience:** Application developers, .NET developers, mobile applications developers, AI developers

**Hands-on Samples:**

- Use ML.NET to classify handwritten digits from zero to nine using the MNIST dataset (bit.ly/2EfpzZ4)
- Comprehensive Cognitive Services SDK sample (bit.ly/30pPTcf)

### Provide Interactive Experience and Insight Through an Enterprise Chatbot

Filing expenses is a mundane task that only gets worse with repetition. Bots powered with AI capabilities can lessen the sting, reducing the time that employees spend inputting data. These bots are created with Azure Bot Service and are Web apps that you can extend using Computer Vision API to accept, analyze and perform OCR on the receipts. You can then store this information to build insights in the longer run.

**Relevant Services:** *Azure Bot Service, Language Understanding Intelligence Service (LUIS), Virtual Assistant, Speech Services, QnA Maker*

Bots powered with Azure Bot Service along with LUIS and the Computer Vision cognitive service can provide a conversational experience to users filing expenses. The addition of the Speech cognitive service allows for building a natural speech-enabled bot that people can use to provide input using voice. You can also explore the open source Virtual Assistant solution to quickly get a fully functional bot running, which you can then customize to your requirements. Finally, the QnA Maker service can be used to provide assistance with first line of questioning coming from the users.

**Target audience:** Application developers, AI developers

**Hands-on Samples:**

- Comprehensive Cognitive Services SDK sample (bit.ly/30pPTcf)
- Quickstart: Recognize speech with the Speech SDK for .NET Core (bit.ly/2VvN7P1)
- Virtual Assistance solution overview (bit.ly/2YyfKgf)
- Quickstart: Create your first Virtual Assistant (C#) (bit.ly/2Hy69zc)
- Quickstart: Create a knowledge base using the QnA Maker API service portal (bit.ly/2EhddzD)

## Wrapping Up

This completes the coverage of the scenarios detailed in **Figure 1** and provides an overview of the spread of Azure AI Services that apply in a typical AI feature integration and development scenario.

If you're looking to add intelligent features in your apps and evaluating AI services available on Azure, you'll need first to decide if you can use any of the Microsoft Cognitive Services, or if you need to build a custom ML model. In many cases, you can use a combination of Cognitive Services and custom ML models to achieve the desired end result.

The pre-built AI services from Microsoft Cognitive Services often meet the requirement for simpler scenarios, such as performing OCR on digital documents, categorizing images based on content, and adding conversational AI in your apps. Some of these services, such as Custom Vision Service and LUIS, also provide extensive customization to meet less-than-simple requirements.

Big Data analytics services such as Azure Databricks and Azure Data Warehouse, along with Azure ML service, can be used for complex scenarios at any scale. Azure ML with automated ML and ML pipelines can increase the productivity of the data science team in an organization by streamlining DevOps processes and reducing time for experimentation.

Now that you've explored Azure AI Services and walked through the expense modernization requirements at Contoso, take a moment to test your understanding by taking the online interactive quiz at ashisa.github.io/AzureAIQuiz. ■

**Ashish Sahu** *is a senior technology evangelist, working with Microsoft India and helping ISVs and startups overcome technical challenges, adopt the latest technologies, and evolve their solutions to the next level.*

**Sagar Bhanudas Joshi** *has been working with developers to help embrace the Universal Windows Platform and the Microsoft Azure platform for more than six years. His current role includes helping SaaS companies architect, design and on-board solutions to Microsoft Cloud Platform, with special focus on ML and AI.*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Sandeep Alur*

# TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

# Training Conference for IT Pros at Microsoft HQ!

## The FUTURE of TECH is HERE

**MICROSOFT HEADQUARTERS**
REDMOND, WA
AUGUST 5-9, 2019

## In-depth Technical Tracks on:

- Client and EndPoint Management
- Cloud
- Infrastructure
- Office 365 for the IT Pro
- PowerShell and DevOps
- Security
- Soft Skills for IT Pros

**SAVE $300**
**WHEN YOU REGISTER BY JULY 12**
Use Promo Code MSDN

EVENT PARTNER Microsoft

SUPPORTED BY Redmond | Redmond Channel Partner | Virtualization & Cloud Review

PRODUCED BY Converge 360 — An 1105 Media company

# AGENDA AT-A-GLANCE

| Client and Endpoint Management | PowerShell and DevOps | Classic Infrastructure | Soft Skills for IT Pros | Security | Azure/ Public Hybrid | Office 365 for the IT Pro |
|---|---|---|---|---|---|---|

| START TIME | END TIME | | | | | |
|---|---|---|---|---|---|---|
| | | **TechMentor Pre-Conference Hands-On Labs: Monday, August 5, 2019** *(Separate entry fee required)* | | | | |
| 7:30 AM | 9:00 AM | Pre-Conference Workshop Registration - Coffee and Light Breakfast | | | | |
| 9:00 AM | 12:00 PM | **HOL01** Hands-On Lab: Building a Bulletproof Privileged Access Workstation (PAW) - *Sami Laiho* | | **HOL02** Hands-On Lab: Mastering Windows Troubleshooting—Advanced Hands-on Workshop - *Bruce Mackenzie-Low* | | |
| 12:00 PM | 2:00 PM | Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center | | | | |
| 2:00 PM | 5:00 PM | Hands-On Lab Continued - *Sami Laiho* | | Hands-On Lab Continued - *Bruce Mackenzie-Low* | | |
| 6:30 PM | 8:30 PM | Dine-A-Round Dinner - Suite in Hyatt Regency Lobby | | | | |
| **START TIME** | **END TIME** | **TechMentor Day 1: Tuesday, August 6, 2019** | | | | |
| 7:00 AM | 8:00 AM | Registration - Coffee and Light Breakfast | | | | |
| 8:00 AM | 9:15 PM | **T01** Microsoft 365 Explained - *Ståle Hansen* | **T02** Top Free Tools for Monitoring Windows Performance - *Bruce Mackenzie-Low* | **T03** Follow the Breadcrumbs Azure Security Center In-Depth - *Mike Nelson* | **T04** From IT Pro to Cloud Pro - *Peter De Tender* | |
| 9:30 AM | 10:45 AM | **T05** Using PowerBI and Azure Log Analytics for End User Devices - *Kevin Kaminski* | **T06** How to Become a Community Rockstar—Learn How to Showcase Your Skills - *Cristal Kawula* | **T07** Backup, Distaster Recovery, and Business Continuity in Azure - *Orin Thomas* | **T08** Regex for Complete Noobs - *Thomas Rayner* | |
| 11:00 AM | 12:00 PM | **Keynote:** Windows Server 2019 Technical Foundation - *Jeff Woolsey, Principal Program Manager, Windows Server/Hybrid Cloud, Microsoft* | | | | |
| 12:00 PM | 1:00 PM | Lunch - McKinley / Visit Exhibitors - Foyer | | | | |
| 1:00 PM | 2:15 PM | **T09** Intune and Custom Policies - *Kevin Kaminski* | **T10** A World Beyond Passwords - *Lesha Bhansali & Kristina Cosgrave* | **T11** Configuring Windows Server & Client for Developing Hosted Linux Workloads - *Orin Thomas* | **T12** To Be Announced | |
| 2:15 PM | 2:45 PM | Sponsored Break - Visit Exhibitors - Foyer | | | | |
| 2:45 PM | 4:00 PM | **T13** OneNote LifeHack: 5 Steps for Succeeding with Personal Productivity - *Ståle Hansen* | **T14** Hardening Windows Server - *Orin Thomas* | **T15** Familiar and Future-Proof: Lift and Shift Your Cluster into Azure IaaS - *Rob Hindman* | **T16** From Cmdlet to Function—Your PowerShell Beginnings - *Mike Nelson* | |
| 4:00 PM | 5:30 PM | Exhibitor Reception – Attend Exhibitor Demo - Foyer | | | | |
| **START TIME** | **END TIME** | **TechMentor Day 2: Wednesday, August 7, 2019** | | | | |
| 7:30 AM | 8:00 AM | Registration - Coffee and Light Breakfast | | | | |
| 8:00 AM | 9:15 AM | **W01** Ten Practical Tips to Secure Your Corporate Data with Microsoft 365 - *Peter Daalmans* | **W02** Surviving a Ransomware Attack: Notes from the Field - *Émile Cabot* | **W03** Everything You Need to Know About Storage Spaces Direct—Part 1 - *Cosmos Darwin* | **W04** Azure is 100 % Highly Available or is it? - *Peter De Tender* | |
| 9:30 AM | 10:45 AM | **W05** Everything You Need to Know About Calling in Microsoft Teams - *Ståle Hansen* | **W06** Get Started with Azure MFA the Right Way - *Jan Ketil Skanke* | **W07** Everything You Need to Know About Storage Spaces Direct—Part 2 - *Cosmos Darwin* | **W08** Become the World's Greatest Azure-Bender - *Peter De Tender* | |
| 11:00 AM | 12:00 PM | **Panel Discussion:** The Future of IT - *Sami Laiho and Dave Kawula (Moderators); Peter De Tender, Thomas Maurer, John O'Neill Sr., and Orin Thomas* | | | | |
| 12:00 PM | 1:00 PM | Birds-of-a-Feather Lunch - *McKinley - Visit Exhibitors - Foyer* | | | | |
| 1:00 PM | 1:30 PM | Networking Break - Exhibitor Raffle @ 1:10 pm (Must be present to win) - *Foyer in front of Business Center* | | | | |
| 1:30 PM | 2:45 PM | **W09** Build Your Azure Infrastructure Like a Pro - *Aleksandar Nikolic* | **W10** Implementing Proactive Security in the Cloud—Part 1 - *Sami Laiho* | **W11** Get the Best of Azure and Windows Server with Windows Admin Center - *Haley Rowland* | **W12** Oww Help! SCCM is Getting SaaSified - *Peter Daalmans* | |
| 3:00 PM | 4:15 PM | **W13** Mastering Azure Using Cloud Shell, PowerShell, and Bash! - *Thomas Maurer* | **W14** Implementing Proactive Security in the Cloud—Part 2 - *Sami Laiho* | **W15** Master Software Definded Networking in Windows Server 2019 - *Greg Cusanza* | **W16** Discussion of Modern Device Management from a (Microsoft Surface) Hardware Engineer - *Carl Luberti* | |
| 5:00 PM | 9:30 PM | TechMentor's Out on the Town in Seattle | | | | |
| **START TIME** | **END TIME** | **TechMentor Day 3: Thursday, August 8, 2019** | | | | |
| 7:30 AM | 8:00 AM | Registration - Coffee and Light Breakfast | | | | |
| 8:00 AM | 9:15 AM | **TH01** Hackers Won't Pass—Microsoft 365 Identity & Threat Protection—Part 1 - *Sergey Chubarov* | **TH02** Azure Cloud Shell: The Easiest Way to Manage Azure with Command-line Tools - *Aleksandar Nikolic* | **TH03** Replicate Your Storage/ Data to Azure the Easy Way - *Arpita Duppala* | **TH04** Android Enterprise Management with Intune - *Jan ketil Skanke* | |
| 9:30 AM | 10:45 AM | **TH05** Hackers Won't Pass—Microsoft 365 Identity & Threat Protection—Part 2 - *Sergey Chubarov* | **TH06** PowerShell Core on Linux: Benefits and Challenges - *Aleksandar Nikolic* | **TH07** A Look Into the Hybrid Cloud Lifestyle of an Azure Stack Operator - *Thomas Maurer* | **TH08** "AaronLocker:" Robust and Practical Application Whitelisting for Windows - *Aaron Margosis* | |
| 11:00 AM | 12:15 PM | **TH09** Using PowerShell to Rock Your Labs - *Dave Kawula* | **TH10** The Force Awakens—Azure SQL Server for the On-prem DBA - *Alexander Arvidsson* | **TH11** 12 Ways to Hack Multi Factor Authentication (MFA) - *Roger Grimes* | **TH12** Windows Autopilot: Modern Device Provisioning - *Michael Niehaus* | |
| 12:15 PM | 2:00 PM | Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center | | | | |
| 2:00 PM | 3:15 PM | **TH13** Migrating to Windows Server 2019 Like the Pro's - *Dave Kawula* | **TH14** Boring is Stable, Stable is Good—Best Practices in Practice for SQL Server - *Alexander Arvidsson* | **TH15** Start Using JEA Today to Stop Overprivileged User Accounts - *John O'Neill Sr.* | **TH16** Wireshark Essentials: Your First Day with Wireshark - *Richard Hicks* | |
| 3:30 PM | 4:45 PM | **TH17** The Case of the Shrinking Data: Data Deduplication in Windows Server 2019 - *Dave Kawula* | **TH18** Malware Protection in Windows 10 - *Émile Cabot* | **TH19** Supporting Surfaces? Learn the Surface Diagnostic Toolkit for Business Now - *John O'Neill Sr.* | **TH20** Always On VPN: The Good, The Bad, and the Ugly! - *Richard Hicks* | |
| **START TIME** | **END TIME** | **TechMentor Post-Conference Workshops: Friday, August 9, 2019** *(Separate entry fee required)* | | | | |
| 8:30 AM | 9:00 AM | Post-Conference Workshop Registration - Coffee and Light Breakfast | | | | |
| 9:00 AM | 12:00 PM | **F01** Workshop: Enhance Security While Increasing Your Admin Superpowers - *John O'Neill Sr.* | | **F02** Workshop: Building Real World Labs in Azure - *Dave Kawula* | | |
| 12:00 PM | 1:00 PM | Lunch - *McKinley* | | | | |
| 1:00 PM | 4:00 PM | Workshop Continued - *John O'Neill Sr.* | | Workshop Continued - *Dave Kawula* | | |

*Logo denotes sessions with a Microsoft speaker.*

*Speakers and sessions subject to change*

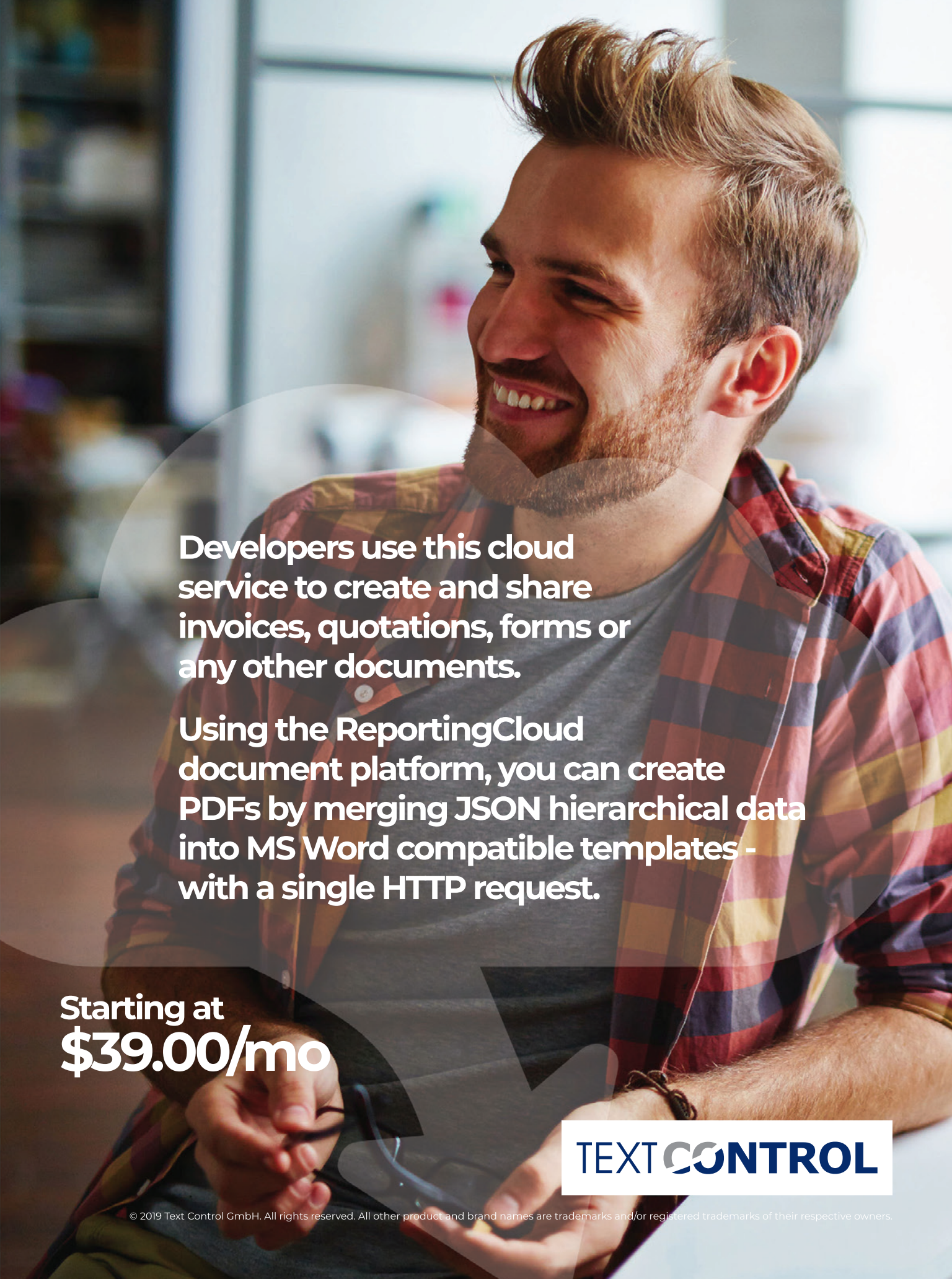
CONNECT WITH TECHMENTOR

Twitter
@TechMentorEvent

Facebook
Search "TechMentor"

LinkedIn
Search "TechMentor"

TechMentorEvents.com/MicrosoftHQ



# Restrict Site Access with AI-Driven Authorization Policies in ASP.NET Core

Stefano Tempesta

ASP.NET Core introduces a claim authorization mechanism that accepts custom policies to restrict access to applications, or to portions of an application, depending on specific authorization attributes of the authenticated user. In my previous article, "AI-Powered Biometric Security in ASP.NET Core" in the June 2019 issue of *MSDN Magazine* (msdn.com/magazine/mt833460), I presented a policy-based model for decoupling authorization logic from the underlying user roles, and showed a specific usage of such authorization policies to restrict physical access to buildings when an unauthorized intrusion is detected. In this second article, I'm going to focus on the connectivity of security cameras and streaming of data to an Azure IoT Hub, triggering the authorization flow, and assessing the severity of a potential intrusion using an anomaly detection service built into Azure Machine Learning.

You'll find more information about the ASP.NET Core authorization framework at bit.ly/2VN9Hmo, and the source code of my Web API at bit.ly/2IXPZCo.

This article discusses:

- Controlling access to buildings using authorization policies
- Registering IoT devices
- Streaming data to the IoT Hub
- Detecting unauthorized intrusions

Technologies discussed:

ASP.NET Core 3, Azure IoT Hub, Azure Machine Learning Studio

Code download available at:

bit.ly/2IXPZCo

## Restricting Access

In my scenario, access to buildings is controlled by authorization policies that have to be met before the doors unlock. ASP.NET Core 3 offers a built-in framework for managing authorization policies, which I leverage in this solution and expose via a Web API. With this in mind, **Figure 1** describes a scenario where a person requests access to a building by swiping an access pass, and cameras detect motion and capture face, body and voice of the person. The card reader and cameras are registered as IoT devices and stream recorded data to Azure IoT Hub.

In my previous article, I described the use of custom authorization policies in ASP.NET Core Web API for checking for specific claims owned by the user. Those claims are asserted as biometric information for face, body and voice, promptly recognized by means of Azure Cognitive Services for Vision and Speech processing.

I'll now describe the process for registering cameras as IoT devices in Azure IoT Hub, and the definition of rules that trigger the authorization flow.

## Registering IoT devices

Generally speaking, IoT applications can be described as things (devices) that send data that generates insights. The insights in turn generate actions to improve a business or process. In my application, an example is the camera (the IoT device) sending image and voice data. This data is used to evaluate whether the person is who they say they are (the insight). The insight is used to authenticate the person and grant them access to the site (the action). The reference architecture design for this solution in **Figure 2** is similar to the recommended Azure IoT architecture (bit.ly/2I20Us2), which describes two ways to process telemetry data: warm path or cold path. The difference has

# 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

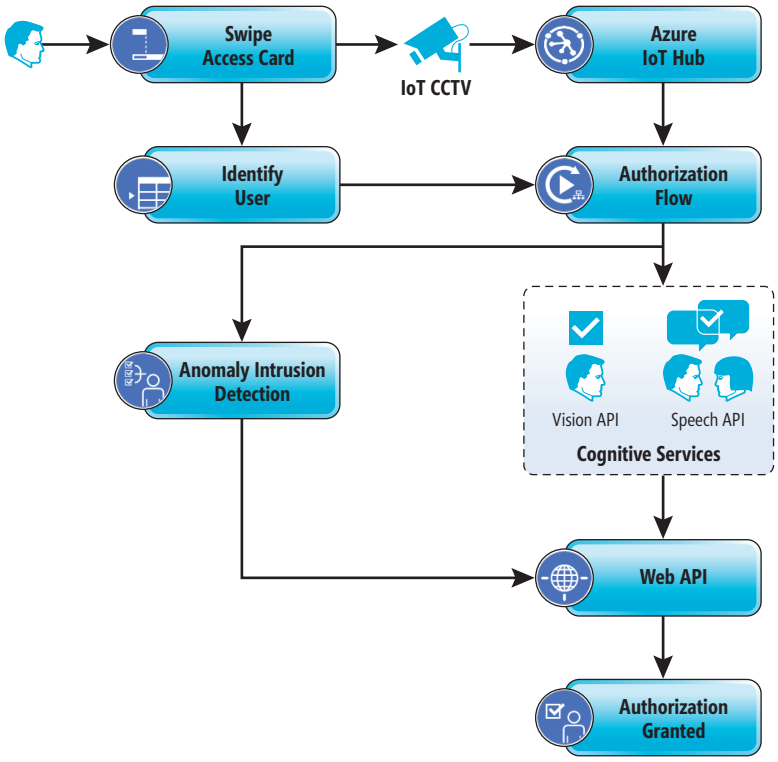



Figure 1 **The Authorization Flow**

to do with requirements for latency and data access. The warm path analyzes data in near-real time, as it arrives, with very low latency. This is the case for triggering vision and speech recognition with Azure Cognitive Services, and then authorizing user access. The cold path captures historical telemetry data for further processing, typically for data analysis or, as in this solution, for machine learning (ML).

The cloud gateway that the registered devices will stream data to is Azure IoT Hub, a managed service, hosted in the cloud, that acts as a central message hub for bi-directional communication between the devices it manages and the authorization application back end. IoT Hub supports communications both from the device to the cloud and from the cloud to the device. It also supports multiple messaging patterns, such as device-to-cloud telemetry, file upload from devices, request-reply methods to control your devices from the cloud, and direct methods, which are cloud-to-device commands that don't require a reply from the device.

A device must be registered with your IoT hub before it can connect. One of the manual processes (there are many!) for device registration is to use Azure Cloud Shell. You first create the device identity by running this command:

```
az iot hub device-identity create --hub-name YourIoTHubName
  --device-id MyDotnetDevice
```

*YourIoTHubName* is the name of the subscriber's IoT Hub in Azure (refer to bit.ly/2JEMnph for a detailed description of how to create an IoT Hub using the Azure Portal), and *MyDotnetDevice* is the name of the device you're registering. After registration, you'll need the device's connection string for streaming data. Run this command to obtain the connection string of the device just registered:

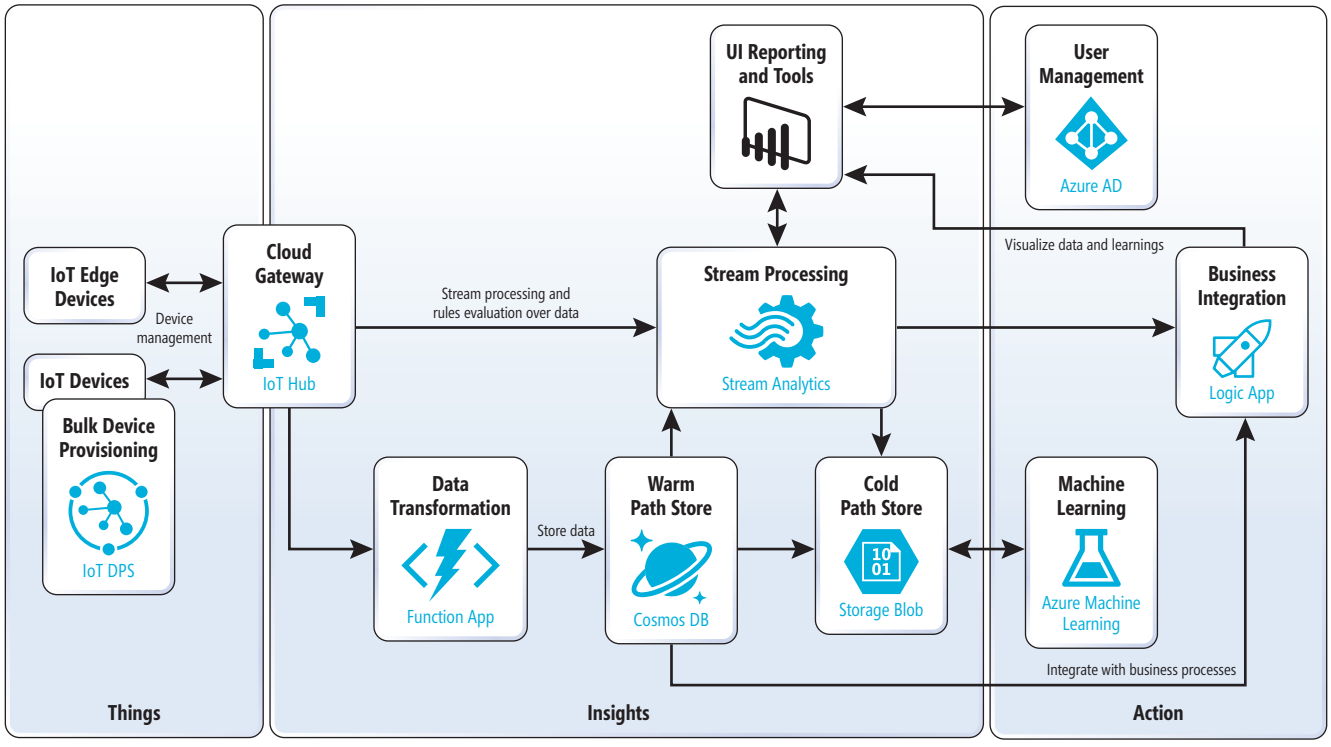



Figure 2 **Azure IoT Reference Architecture**

```
az iot hub device-identity show-connection-string --hub-name YourIoTHubName
  --device-id MyDotnetDevice --output table
```

Make a note of the device connection string, which should look something like:

```
HostName={YourIoTHubName}.azure-devices.net; DeviceId=MyNodeDevice;
  SharedAccessKey={YourSharedAccessKey}
```

You'll also need the Event Hubs-compatible endpoint, Event Hubs-compatible path, and the primary key from your access policy to enable the back-end application to connect to your IoT hub and retrieve the messages. Although IoT Hub has a predefined "iothubowner" access policy, with full control on the hub, it's recommended you create a less-privileged policy for device connection. The following commands retrieve these values for your IoT hub:

```
az iot hub show --query properties.eventHubEndpoints.events.endpoint
  --name YourIoTHubName

az iot hub show --query properties.eventHubEndpoints.events.path
  --name YourIoTHubName

az iot hub policy show --name YourAccessPolicy --query primaryKey
  --hub-name YourIoTHubName
```

You can also enroll a device in an IoT hub using the Azure IoT Hub Provisioning Service Client in .NET. You'll need the Microsoft.Azure.Devices.Provisioning.Service package as a prerequisite, which you can install in your Visual Studio solution from NuGet.

The DeviceRegistrationAsync method in **Figure 3** registers a Trusted Platform Module (TPM)-based device using the device's endorsement key, which is permanently embedded in the hardware, generally at the time of manufacture. It also requires a registration ID that's used to uniquely identify a device. This may or may not be the same as the device ID. For TPM-based devices, the registration ID may be derived from the TPM itself, for example, an SHA-256 hash of the TPM endorsement key.

Querying the enrollmentResult.RegistrationState will indicate the registration status of the device.

## Streaming Data to the IoT Hub

Once a device is successfully registered, it can start streaming data to the IoT Hub. IoT Hub supports the HTTPS and Advanced Message Queuing Protocol (AMQP) protocols for data ingestion, and it's format-agnostic, meaning that the data format can be anything, really, from CS, to JSON to Avro (an Apache data serialization project: avro.apache.org). If you're designing a connection of devices from remote locations, where a small code footprint is required or the network bandwidth is limited, you may want to consider Message Queuing Telemetry Transport (MQTT: mqtt.org), a lightweight messaging protocol for small sensors and mobile devices, optimized for high-latency or unreliable networks. There's a 256K limit in a device-to-cloud message, though, which makes direct data streaming unpractical for capturing image and voice data. Another approach to data loading that IoT Hub supports is file upload to blob. A camera first records audio and video at the edge, that is, on the device itself, and then uploads this data to the IoT Hub. When the upload is complete, IoT Hub raises a file upload notification message through a service-facing endpoint. This event then triggers the authorization process that, eventually, invokes the Web API with the ASP.NET Core authorization policies. It's important to note that the file upload mechanism requires an Azure Blob storage account. Messages aren't brokered through IoT Hub itself. Instead, IoT Hub acts as a dispatcher to an associated storage account, so it's obviously important to configure the storage account in Azure and associate it with your IoT hub. For detailed instructions, see bit.ly/2YOMz8Q.

Figure 3 **The DeviceRegistrationAsync Method**

```
using Microsoft.Azure.Devices.Provisioning.Service;
static async Task DeviceRegistrationAsync()
{
  Attestation attestation = new TpmAttestation("<TpmEndorsementKey>");

  IndividualEnrollment enrollment = new IndividualEnrollment(
    "<RegistrationId>", attestation)
  {
    DeviceId = "DeviceId"
  };

  var serviceClient = ProvisioningServiceClient.CreateFromConnectionString(
    "<ConnectionString>");

  IndividualEnrollment enrollmentResult = await
    serviceClient.CreateOrUpdateIndividualEnrollmentAsync(enrollment)
      .ConfigureAwait(false);
}
```

To initialize a file upload process, a device sends a POST request to an endpoint on the IoT Hub with this format:

```
{iot hub}.azure-devices.net/devices/{deviceId}/files
```

The POST request contains this JSON body:

```
{
  "blobName": "..."
}
```

IoT Hub returns a JSON response, which the device uses to upload the file:

```
{
  "correlationId": "<correlation_id>",
  "hostName": "<yourstorageaccount>.blob.core.windows.net",
  "containerName": "<container_name>",
  "blobName": "<blob_name>",
  "sasToken": "<token>"
}
```

When the upload is complete, the device sends a POST request to:

```
{iot hub}.azure-devices.net/devices/{deviceId}/files/notifications

with the following JSON body:
{
  "correlationId": "<correlation ID received from the initial request>",
  "isSuccess": bool,
  "statusCode": XXX,
  "statusDescription": "Description of status"
}
```

If you're more into C#, you can upload a file by setting up a device client from the device connection string and send it as a stream to the blob asynchronously:

```
async void SendToBlobAsync(string blobName, Stream stream)
{
  using (DeviceClient deviceClient =
    DeviceClient.CreateFromConnectionString(
      "<ConnectionString>", TransportType.Http1))
  {
    await deviceClient.UploadToBlobAsync(blobName, stream);
  }
}
```

## Intrusion Detection

All the data-related activities described so far happen on what's called the "warm path store," as data is processed in near-real time. Telemetry data is also archived in Azure Blob storage persistently for further analysis. This is the "cold path store" used by Azure Machine Learning Studio as a data source for training a data model and detecting unauthorized intrusion.

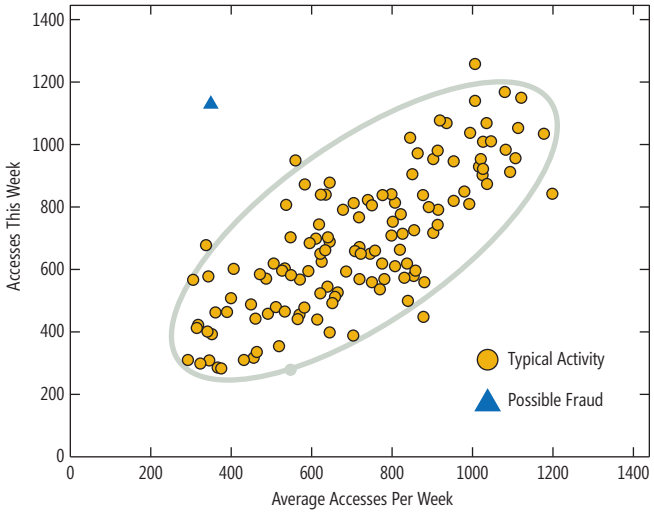



Figure 4 **Anomalous Intrusion Representation**

If there's a mismatch between the detected identity of the person and their access pass, access to the site is blocked immediately. Otherwise, the flow carries on by checking whether any of the following anomalies have been encountered:

- Atypical frequency of access to the building.
- Whether the person has exited the building earlier (check out).
- Number of accesses permitted per day.
- Whether the person is on duty.
- Criticality of the building (you may not want to restrict access to a canteen, but enforce a stricter policy for access to a server datacenter).
- Whether the person is bringing someone else or something else along.
- Past occurrences of similar access typologies to the same building.
- Risk-level changes measured in the past.
- Number of intrusions detected in the past.

The anomaly detection service runs in Azure ML and returns a score, expressed as a likelihood that the access is a deviation from the standard, or not. The score is expressed in a range between zero and one, where zero is "no risk detected," all good, full trust granted; and one is "red alert," block access immediately! The risk level of each building determines the threshold that's considered acceptable for allowing access into the building for any value greater than zero.

Intrusion detection is an anomaly detection problem that identifies and predicts rare or unusual occurrences of data points in a cluster. ML anomaly detection has been broadly adopted for detecting potential payment frauds, catching abnormal device readings, discovering network intrusion and any application where there's a need to explore a large dataset of unpredictable activities or elements.

For example, access to sites can be registered over time and grouped by different criteria (time of day, a person's role, whether solo or escorted, previous access, and so forth). These criteria are called "features" and identify all the conditions that the ML algorithm will verify to assess the intrusion score, including whether it's a new access and whether a data point sits within the average recorded values—a cluster—or outside. If the data point is outside, its distance from the border—deviation—is also measured, to indicate a lower or higher risk factor as a number between zero and one (see **Figure 4**).

There can be hundreds of features and each contributes, to varying extents, toward the intrusion probability. The degree to which each feature contributes to the intrusion score isn't determined by a person, say the Head of Security, but is inferred by the ML process as part of the model training. So, with regard to the unauthorized building access, if the use of someone else's pass to gain access to a building is proven to have high occurrences, the weight of such typology of intrusion will be equally so. And, if this practice diminishes, the contribution level would decrease in parallel. Simply put, these models self-learn without explicit programming, such as with a manual review.

The process of analyzing data to extrapolate a meaning is called feature engineering; in the field of anomaly detection, you typically want to look at:

- Aggregated variables: the total number of transactions per location in the last 24 hours, seven days, or 30 days, for example.
- Mismatch values: any mismatches between the biometric information of the user and the access pass, or detection of someone's presence in multiple places at the same time or with too short a time difference between two places very distant from each other.
- Risk tables: intrusion risks calculated using historical probability, grouped by site, level of access restriction to a building and so on.

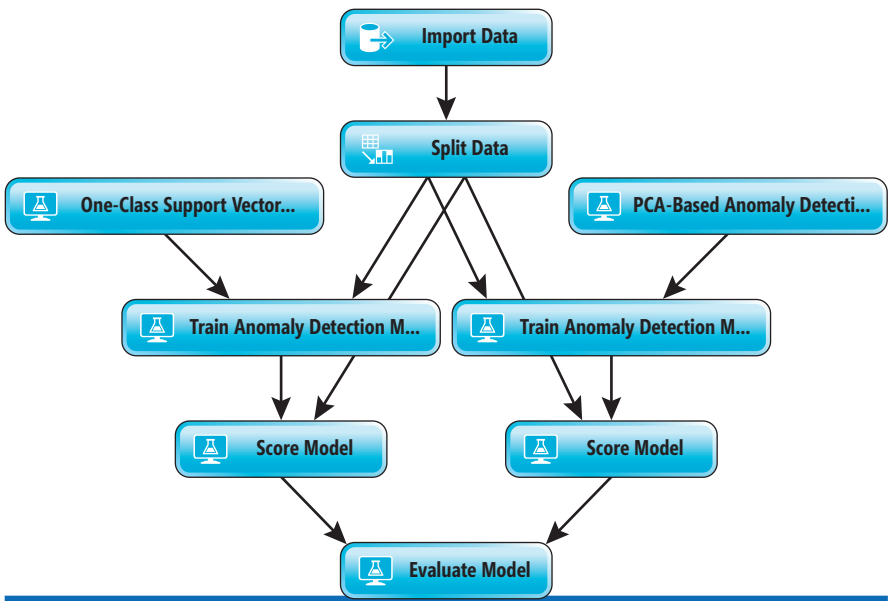



Figure 5 **Site Intrusion Detection Experiment in Azure Machine Learning Studio**

## Azure Machine Learning Studio

Azure ML Studio provides a visual editor for building ML experiments starting from a dataset, and then executing model training, scoring and evaluation. But let's go in order. **Figure 5** shows the complete ML flow.

The first step is importing the dataset. As data is stored in an Azure Blob, Azure ML Studio provides a specific module called "Import Data" that you can use for connecting to an Azure Blob service. After importing data, you need to separate it into training and testing sets, using the "Split Data" module. You can choose different splitting modes, depending on the type of data you have, and how you want to divide it. For this solution, I've selected the Split Rows option to divide the data into two random parts, with 80 percent of the data assigned to the training dataset and the remaining to be used for testing. The ML flow then performs a training of the dataset. Anomaly detection is a classification problem that can be executed as supervised or unsupervised learning, using either of the following approaches:

- One-Class Support Vector Model
- Principal Component Analysis

You can use the One-Class Support Vector Model module to create an anomaly detection model that's particularly useful in scenarios where the data is mostly "normal" data, without many cases of the anomalies you're trying to detect. For example, you might need to detect fraudulent accesses, but not have many examples of fraud you can use to train a typical classification model, but you might have many examples of good cases.

Principal Component Analysis, which is frequently abbreviated to PCA, in contrast, is an established ML technique that can be applied to exploratory data analysis when the source dataset isn't well known. PCA reveals the inner structure of the data and explains the variance in the data by analyzing multiple variables and the possible correlation and combination of values that best captures the differences in outcomes. These values are called "principal components" as they're the key factors that influence outcome.

As at this stage I can't anticipate which approach works better, I'll use both, with two separate Train Anomaly Detection Model modules, then compare the reciprocal results with an evaluation of the predicted values. The "Score Model" module scores predictions for a trained model, and the "Evaluate Model," as the name implies, evaluates the results of the classification model with standard metrics, such as accuracy (goodness of a classification model as the proportion of true results to total cases), precision (proportion of true results over all positive results) and recall (fraction of all correct results returned by the model). The scored dataset with the higher metrics would be the preferred one for producing the predictive service associated with this training experiment.

Azure ML Studio generates a Web service from a predictive experiment and exposes it as a REST API that external applications can consume. The API has an endpoint hosted on the services.azureml.net domain, which is unique to your service. It requires authentication with an API key passed in the HTTP request header as the Authorization:Bearer property. The content type of the request is application/json, and the request body assumes the form of a JSON payload that contains the input values for the predictive service. The output of the service is also a JSON response with the scored value. The C# code in **Figure 6** shows how to consume the ML service with an HTTP client. After building the request as a collection of string arrays, the HTTP client is initialized with the API key in the request header's authorization property, and its base address is set to the URI of the Web service. The request is submitted by POST as a JSON message, asynchronously.

Figure 6 **Invoking the Azure ML Web Service**

```
async Task InstrusionDetectionAsync()
{
  using (var client = new HttpClient())
  {
    var scoreRequest = new
    {
      Inputs = new Dictionary<string, StringTable>
      {
        ["input1"] = new StringTable
        {
          ColumnNames = new [] { "<Feature1>", "<Feature2>", "<Feature3>" },
          Values = new [,] { { "<Value1>", "<Value2>", "<Value3>" } }
        }
      }
    };

    client.DefaultRequestHeaders.Authorization =
      new AuthenticationHeaderValue("Bearer", "<ApiKey>");
    client.BaseAddress = new Uri("<EndpointURI>");

    HttpResponseMessage response = await client.PostAsJsonAsync("", scoreRequest);
    if (response.IsSuccessStatusCode)
    {
      string result = await response.Content.ReadAsStringAsync();
    }
  }
}

public class StringTable
{
  public string[] ColumnNames { get; set; }
  public string[,] Values { get; set; }
}
```

## Wrapping Up

The Azure cloud offers a rich toolset of resources that, in the right combination, help build end-to-end solutions that integrate IoT, Machine Learning, Cognitive Services and ASP.NET Core API. The scenario illustrated in the previous article of this two-parter exposes the richness of the custom policy framework in .NET Core for user authorization, in synergy with the Vision and Speech APIs of Cognitive Service for recognition of biometric characteristics such as face and voice. The current article focuses on collecting such biometric information from cameras registered as IoT devices, and streaming data to an IoT Hub in Azure. The solution is enriched with an ML service that supports the authorization process with the analysis of the access request against an historical dataset, for detecting a potential unauthorized intrusion. ■

**Stefano Tempesta** *is a Microsoft Regional Director, MVP on AI and Business Applications, and member of Blockchain Council. A regular speaker at international IT conferences, including Microsoft Ignite and Tech Summit, Tempesta's interests extend to blockchain and AI-related technologies. He created Blogchain Space (blogchain.space), a blog about blockchain technologies, writes for* MSDN Magazine *and* MS Dynamics World, *and publishes machine learning experiments on the Azure AI Gallery (gallery.azure.ai).*

**Thanks** to the following Microsoft technical expert for reviewing this article: *Danilo Diaz*

Visual Studio LIVE!
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

# MICROSOFT HQ

August 12-16, 2019
Microsoft Campus in Redmond, WA


## 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

**Save $300**
**When You Register By July 19!**
Use Promo Code MSDN

vslive.com/microsofthq

GOLD SPONSOR
Progress

SUPPORTED BY
Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY
CONVERGE 360

# AGENDA AT-A-GLANCE

#VSLIVE

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

| START TIME | END TIME | | | | |
|---|---|---|---|---|---|
| | | **Pre-Conference Full Day Hands-On Labs: Monday, August 12, 2019** *(Separate entry fee required)* | | | |
| 7:00 AM | 8:30 AM | Pre-Conference Workshop Registration - Coffee and Morning Pastries | | | |
| 8:30 AM | 12:30 PM | **HOL01** Hands-On Lab: Hands-On With Cloud-Native .NET Development - *Rockford Lhotka and Richard Seroter* | **HOL02** Hands-On Lab: Leveling Up—Dependency Injection in .NET - *Jeremy Clark* | **HOL03** Full Day Hands-On Lab: Xamarin and Azure: Build the Mobile Apps of Tomorrow - *Laurent Bugnion* | |
| 12:30 PM | 2:00 PM | Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center | | | |
| 2:00 PM | 5:30 PM | **HOL01** Hands-On Lab Continued - *Rockford Lhotka and Richard Seroter* | **HOL02** Hands-On Lab Continued - *Jeremy Clark* | **HOL03** Hands-On Lab Continued - *Laurent Bugnion* | |
| 7:00 PM | 9:00 PM | Dine-A-Round Dinner | | | |
| **START TIME** | **END TIME** | **Visual Studio Live! Day 1: Tuesday, August 13, 2019** | | | |
| 7:00 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **T01** Angular 101 - *Deborah Kurata* | **T02** Stunning Mobile Apps with the Xamarin Visual Studio System - *James Montemagno* | **T03** SQL Server 2019 Deep Dive - *Scott Klein* | **T04** C# and/or .NET Core 3.0 - *Dustin Campbell* |
| 9:30 AM | 10:45 AM | **T05** Moving to ASP.NET Core 2.X - *Philip Japikse* | **T06** Sharing C# Code Across Platforms - *Rockford Lhotka* | **T07** Application Modernization with Microsoft Azure - *Michael Crump* | **T08** DevOps Practices Can Make You A Better Developer - *Robert Green* |
| 10:45 AM | 11:15 AM | Sponsored Break - Visit Exhibitors - *Foyer* | | | |
| 11:15 AM | 12:15 PM | **KEYNOTE: Moving .NET Beyond Windows** - *James Montemagno, Principal Program Manager - Mobile Developer Tools, Microsoft* | | | |
| 12:15 PM | 1:30 PM | Lunch - McKinley / Visit Exhibitors - *Foyer* | | | |
| 1:30 PM | 2:45 PM | **T09** Angular Best Practices - *Deborah Kurata* | **T10** To Be Announced | **T11** Data Pipelines with End-2-End Azure Analytics - *Scott Klein* | **T12** Get Func-y: Understanding Delegates in .NET - *Jeremy Clark* |
| 3:00 PM | 4:15 PM | **T13** Versioning ASP.NET Core APIs -*Philip Japikse* | **T14** Visual Studio for Mac: From Check-out to Publish - *Dominic Nahous and Cody Beyer* | **T15** Why, When, and How to Enhance Your App with Azure Functions - *Daria Grigoriu & Eamon O'Reilly* | **T16** CI / CD with Azure DevOps - *Tiago Pascoal* |
| 4:15 PM | 5:45 PM | Microsoft Ask the Experts & Welcome Reception | | | |
| **START TIME** | **END TIME** | **Visual Studio Live! Day 2: Wednesday, August 14, 2019** | | | |
| 7:30 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **W01** Getting Pushy with SignalR and Reactive Extensions - *Jim Wooley* | **W02** AI and Analytics with Apache Spark on Azure Databricks - *Andrew Brust* | **W03** Building Business Application Bots - *Michael Washington* | **W04** DevOps with ASP.NET Core, EF Core, & Azure DevOps - *Benjamin Day* |
| 9:30 AM | 10:45 AM | **W05** Building Reactive Client Experiences with RxJs and Angular Forms *Jim Wooley* | **W06** Power BI: What Have You Done for Me Lately - *Andrew Brust* | **W07** Windows Subsystem for Linux - *Rich Turner and Craig Loewen* | **W08** Real World Scrum with Azure DevOps - *Benjamin Day* |
| 11:00 AM | 12:00 PM | **GENERAL SESSION:** .NET Core 3 - *Scott Hunter, Director of Program Management, Microsoft* | | | |
| 12:00 PM | 1:30 PM | Birds-of-a-Feather Lunch - McKinley / Visit Exhibitors - Foyer | | | |
| 1:30 PM | 2:45 PM | **W09** C# in the Browser with Client-side Blazor and WebAssembly - *Rockford Lhotka* | **W10** UX Design Fundamentals: What do Your Users Really See? - *Billy Hollis* | **W11** Works On My Machine... Docker for Developers - *Chris Klug* | **W12** Bolting Security Into Your Development Process - *Tiago Pascoal* |
| 2:45 PM | 3:15 PM | Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win) - Foyer | | | |
| 3:15 PM | 4:30 PM | **W13** Creating Business Applications Using Blazor (Razor Components) - *Michael Washington* | **W14** What's New with Azure App Service? - *Christina Compy* | **W15** Microservices with Azure Kubernetes Service (AKS) - *Vishwas Lele* | **W16** Database DevOps with SQL Server - *Brian Randell* |
| 5:15 PM | 9:30 PM | VSLive!'s Out On the Town in Seattle | | | |
| **START TIME** | **END TIME** | **Visual Studio Live! Day 3: Thursday, August 15, 2019** | | | |
| 7:30 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **TH01** Angular Application Testing Outside the Church of TDD - *Chris Klug* | **TH02** Add Native to Cross Platform with Xamarin.Essentials - *Veronika Kolesnikova* | **TH03** Best Practices for Serverless DevOps: Inner Loop, Outer Loop, Observability - *Colby Tresness* | **TH04** Visual Studio Productivity Tips and Tricks - *Allison Buchholtz-Au* |
| 9:30 AM | 10:45 AM | **TH05** Any App, Any Language with Visual Studio Code and Azure DevOps - *Brian Randell* | **TH06** MVVM Made Easy for WPF Applications - *Paul Sheriff* | **TH07** Cross Platform Automation with PowerShell Core 6.0 | **TH08** Exceptional Development: Dealing With Exceptions in .NET - *Jason Bock* |
| 11:00 AM | 12:15 PM | **TH09** To Be Announced | **TH10** Now, The Two Worlds Collided - *Veronika Kolesnikov* | **TH11** What Every Developer Needs to Know About Deep Learning - *Vishwas Lele* | **TH12** Use Async Internals in .NET - *Adam Furmanek* |
| 12:15 PM | 2:00 PM | Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center | | | |
| 2:00 PM | 3:15 PM | **TH13** What's New in Bootstrap 4 - *Paul Sheriff* | **TH14** How to Interview a Developer - *Billy Hollis* | **TH15** Building Azure Cosmos DB Applications – Part I - *Leonard Lobel* | **TH16** Testability in .NET - *Jason Bock* |
| 3:30 PM | 4:45 PM | **TH17** To Be Announced | **TH18** To Be Announced | **TH19** Building Azure Cosmos DB Applications – Part II - *Leonard Lobel* | **TH20** Manual Memory Management in .NET Framework - *Adam Furmanek* |
| **START TIME** | **END TIME** | **Post-Conference Workshops: Friday, August 16, 2019** *(Separate entry fee required)* | | | |
| 7:30 AM | 8:00 AM | Post-Conference Workshop Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 5:00 PM | **F01** Workshop: ASP.NET Core with Azure DevOps - *Brian Randell* | **F02** Workshop: Web Development in 2019 - *Chris Klug* | **F03** Workshop: SQL Server for Developers: The Grand Expedition - *Andrew Brust and Leonard Lobel* | |

*Speakers and sessions subject to change*

*Denotes Microsoft Speaker/Session*

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

# Create a Machine Learning Prediction System Using AutoML

James McCaffrey

Microsoft ML.NET is a large, open source library of machine learning functions that allows you to create a prediction model using a C# language program, typically in Visual Studio. Writing a program that directly uses ML.NET to create a prediction model isn't simple. The AutoML system uses the ML.NET command-line interface (CLI) tool to automatically create a prediction model for you, and also generates sample code that uses the model, which you can then customize.

A good way to understand what the ML.NET CLI and AutoML are, and to see where this article is headed, is to examine the screenshot of a demo system in **Figure 1**. The demo uses a file of training data named people_train.tsv to create a prediction model, and a file of test data named people_test.tsv to evaluate the accuracy of the prediction model. The goal of the demo is to predict the political leaning of a person (conservative, moderate, liberal) from their age, sex, geo-region and annual income.

The demo program runs AutoML in a Windows CMD shell by executing the command:

```
mlnet auto-train ^
--task multiclass-classification ^
--dataset ".\Data\people_train.tsv" ^
--test-dataset ".\Data\people_test.tsv" ^
--label-column-name politic ^
--max-exploration-time 5
```

The carat character is used for line continuation in a command shell. The same AutoML command can be run in PowerShell, too, by using the backtick line continuation character instead of the carat.

AutoML automatically creates and evaluates several different machine learning models using different algorithms, such as SgdCalibratedOva ("stochastic gradient descent calibrated one versus all") and AveragedPerceptronOva. For the demo run, AutoML identified the LightGbmMulti ("lightweight gradient boosting machine multiclass") algorithm as the best option, with a prediction accuracy of 77.01 percent on the test data.

After examining the different prediction models, AutoML saved the best model as MLModel.zip and also generated sample C# code and a Visual Studio project file named the rather lengthy SampleMulticlassClassification.Console.App.csproj to use for the model.

Some of the technologies discussed in this article are still in preview. All information is subject to change.

**This article discusses:**

- Using AutoML to create a prediction model
- Understanding the data
- Multiclass classification
- Interpreting the results
- Using the generated model

**Technologies discussed:**

Microsoft ML.NET, AutoML, .NET Core SDK, Visual Studio, C#

**Code download available at:**

msdn.com/magazine/0719magcode

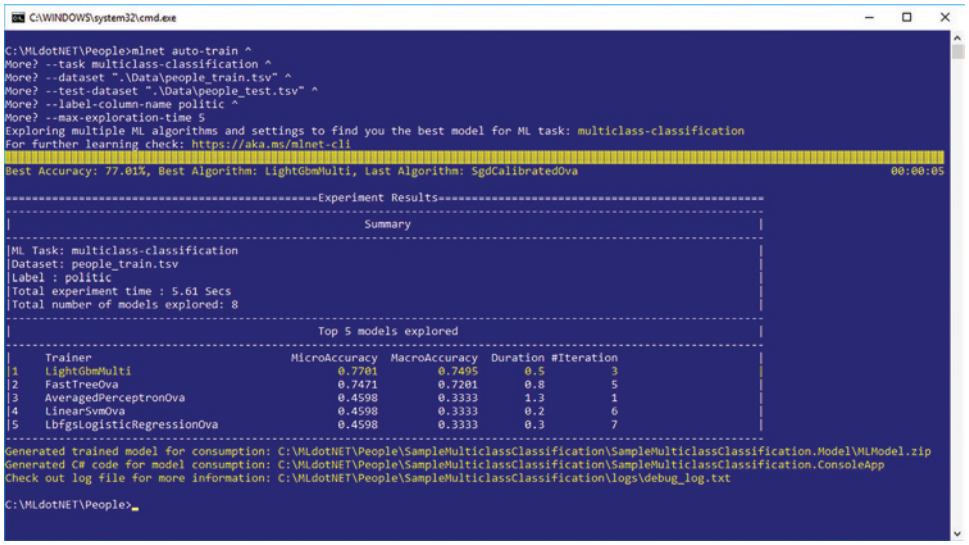



Figure 1 **AutoML and ML.NET CLI in Action**

**Figure 2** shows an example of how the trained model can be used from within a C# program. The generated code was edited to make a prediction for a person who is 33 years old, male, lives in the "central" region and has a $62,000.00 annual income. The prediction is that the person is a political "moderate."

The demo program also displays the prediction probabilities for conservative, moderate, liberal: (0.0034, 0.9055, 0.0912). These aren't true probabilities in the mathematical sense, but they do give you a loose suggestion of the confidence of the prediction. In this case, the model seems quite certain (p = 0.9055) that the person is "moderate."

Both ML.NET CLI and AutoML are currently in Preview mode and are in rapid development, so some of the information presented here may have changed by the time you read this article. However, most of the changes should be in the form of additional features rather than in underlying architecture.

This article assumes you have intermediate or better skills with C#, and basic familiarity with working in a command shell, but doesn't assume you know anything about ML.NET CLI or AutoML. All the demo code is presented in this article. The two demo data files are available in the download that accompanies this article.

## Understanding the Data

Most machine learning problems start with analysis and preparation of the available data, and that's the case when using ML.NET CLI and AutoML. The training data has 1,000 items and looks like:

```
sex   age  region   income    politic
False  26  eastern  53800.00  conservative
False  19  western  39200.00  moderate
True   19  central  80800.00  liberal
False  52  eastern  86700.00  conservative
False  56  eastern  89200.00  liberal
...
```

The test data has the same format and consists of 200 items. The data is synthetic and was generated programmatically. Both files are tab-delimited and have a .tsv extension to identify them as such to AutoML. AutoML also supports space-delimited (.txt) and comma-delimited (.csv) files. AutoML supports data files with or without a header line, but, as you'll see shortly, supplying a header line is more convenient than not supplying one.

Although there are many kinds of prediction problems, there are three fundamental types: multiclass classification, binary classification and regression. The goal of a multiclass classification problem is to predict a discrete value where there are three or more possible values to consider. For example, predicting the political leaning (conservative, moderate, liberal) of a person based on their age, sex, geo-region and annual income, as in the demo program.

The goal of a binary classification problem is to predict a discrete value that can be one of just two possible values. For example, you might want to predict the sex (male or female) of a person based on their age, geo-region, income and political leaning. If you're new to machine learning, you might

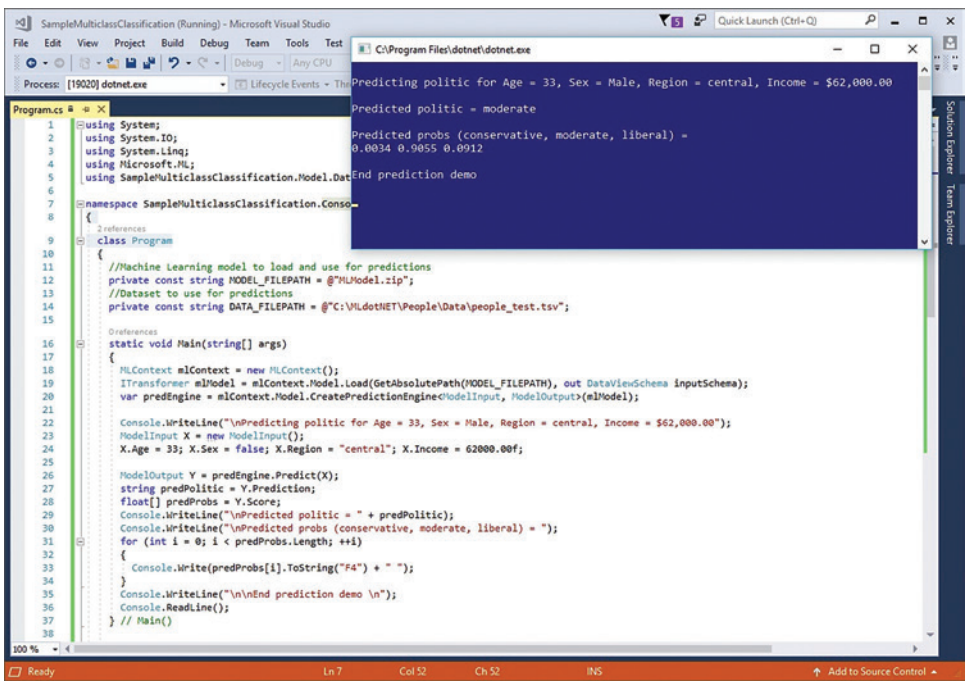



Figure 2 **Using the AutoML Model to Make a Prediction**

find it a bit strange that binary classification and multiclass classification are considered different categories. It turns out that the two types of problems have some fundamental math differences.

The goal of a regression problem is to predict a single numeric value. For example, you might want to predict the annual income of a person based on their age, sex, geo-region and political leaning. AutoML currently supports multiclass classification, binary classification and regression. Support will eventually be added for other types of problems, such as ranking and clustering.

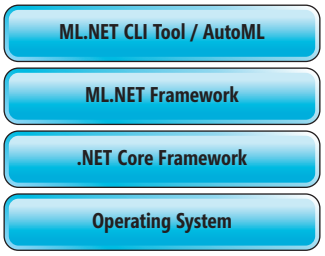



Figure 3 **AutoML Components**

The demo data files illustrate the use of binary, integer, categorical and floating-point data. When working with binary data, such as the sex variable, you should use True and False (either uppercase or lowercase), rather than using 0 and 1. The demo uses False for male and True for female, so you can think of the sex variable as "is-female."

It's important to keep AutoML files and directories organized. I created a top-level directory named MLdotNET. Then, within MLdotNET, I created a directory named People to act as the root for AutoML files associated with the people data. Within the People directory I created a directory named Data and placed files people_train.tsv and people_test.tsv there. I executed AutoML commands while in the root People directory because AutoML generates subdirectories inside the root directory from which commands are issued.

## Installing AutoML

As is often the case with pre-release software, I ran into several minor glitches while installing AutoML and you can expect a few hiccups, too. Briefly, there are three steps to getting AutoML up and running. First, install Visual Studio if necessary. Second, install the .NET Core SDK if necessary. Third, install the ML.NET CLI tool that contains AutoML.

It's possible to use AutoML without Visual Studio, but the models created by AutoML are designed specifically for Visual Studio. I successfully used Visual Studio 2017 Professional and the free Visual Studio 2017 Community edition. The AutoML documentation states that AutoML works with Visual Studio 2019, but I was unable to get my models to load using it.

The AutoML system relies on the .NET Core framework, in particular the .NET Core SDK. After a bit of trial and error, I succeeded by installing SDK version 2.2.101. The installation process uses a standard self-extracting executable with a nice GUI. Installing the .NET Core SDK also gives you the Runtime environment, so you don't have to install it separately.

After installing the .NET Core SDK, the last step is to install AutoML. AutoML isn't a standalone program, instead it resides within a tool named mlnet, which is a bit confusing. To install the mlnet tool, you launch a shell (CMD or PowerShell) and issue the command > dotnet tool install -g mlnet. The command reaches out to the Internet (so you must be online) to a default repository and installs mlnet to a default location on your machine. After a couple of false starts I eventually got a "Tool 'mlnet' (version 0.3.0) was successfully installed" message. I verified the installation by issuing the command > dotnet tool list -g.

The diagram in **Figure 3** shows the relationship between the key components used in an AutoML system.

## Multiclass Classification

To create a multiclass classification problem using AutoML, a minimum of three arguments are needed, for example:

```
mlnet auto-train ^
--task multiclass-classification ^
--dataset ".\Data\people_train.tsv" ^
--label-column-name politic ^
```

The task type can be "multiclass classification." "binary classification" or "regression." The dataset argument specifies the path to the training data. You can use either Windows-style backslash characters or Linux-style forward slashes. The test-dataset argument is optional. If test-dataset isn't present, AutoML will evaluate the trained model using the training data.

You can supply an optional validation-dataset argument to allow AutoML to use the train-validate-test paradigm. During training, AutoML monitors the error associated with the model on the validation data, and when the error starts to increase, training can stop early so that the model doesn't become overfitted to the training data.

The label-column-name argument specifies the name of the column that contains the variable to predict. If your training dataset doesn't have a header, you can use the label-column-index argument with a 1-based column index, for example --label-column-index 5.

Figure 4 **AutoML Command Summary**

| Argument | Alias | Values | Default Value |
|---|---|---|---|
| --task | -T | multiclass classification, binary classification, regression | |
| --dataset | -d | *path to file* | |
| --test-dataset | -t | *path to file* | *none* |
| --validation-dataset | -v | *path to file* | *none* |
| --label-column-name | -n | *column name in header of variable to predict* | |
| --label-column-index | -i | *1-based column index in header of variable to predict* | |
| --ignore-columns | -I | *comma-separated column names in header to ignore* | none |
| --has-header | -h | true, false | true |
| --max-exploration-time | -x | *time in seconds* | 10 |
| --verbosity | -V | quiet, minimal, diagnostic | minimal |
| --cache | -c | on, off, auto | auto |
| --name | -N | *name of created output project* | Sample{task} |
| --output-path | -o | *directory to place output project* | *current directory* |
| --help | -h | | |

The table in **Figure 4** summarizes the 14 arguments for AutoML. Each argument has a full name preceded by double hyphens and a shortcut alias of one case-sensitive letter preceded by a single hyphen. The meaning of most of the arguments is self-explanatory. The --cache argument tells AutoML to load all data into memory if possible (on) or not (off) or to automatically determine what to do.

The --max-exploration-time argument has the biggest impact on AutoML results. In general, the more time you allow AutoML to work, the better the prediction model that will be generated. AutoML does what's in effect a double exploration. First, it tries different algorithms that are applicable to the type of prediction task. For example, for a multiclass classification problem, AutoML currently supports 10 algorithms: AveragedPerceptronOva, FastForestOva, FastTreeOva, LbfgsLogisticRegressionOva, LbfgsMaximumEntropyMulti, LightGbmMulti, LinearSvmOva, SdcaMaximumEntropyMulti, SgdCalibratedOva, SymbolicSgdLogisticRegressionOva.

Second, for each applicable algorithm, AutoML tries different values of the hyperparameters that are specific to the algorithm. For example, the FastTreeOva algorithm requires you to specify values for five parameters: NumberOfLeaves, MinimumExampleCountPerLeaf, NumberOfTrees, LearningRate and Shrinkage. The LightGbmMulti algorithm requires values for 13 parameters, including items such as NumberOfIterations, LearningRate and L2Regularization.

The number of different combinations of algorithms and hyperparameters is unimaginably large. Human machine learning experts rely on intuition and experience to find a good algorithm and a good set of hyperparameters, but the process is extremely tedious and time-consuming. AutoML does a sophisticated search automatically.

## Interpreting the Results

If you refer to the screenshot in **Figure 1**, you'll see that AutoML displays this information for the five best models found:

```
    Trainer               MicroAccuracy  MacroAccuracy
1   LightGbmMulti         0.7701         0.7495
2   FastTreeOva           0.7471         0.7201
3   FastForestOva         0.7471         0.7236
4   AveragedPerceptronOva 0.4598         0.3333
5   LinearSvmOva          0.4598         0.3333
```

The MicroAccuracy and MacroAccuracy values give you two different metrics for model prediction accuracy. MicroAccruacy is the more important of the two. MicroAccuracy is normal accuracy—just the number of correct predictions on the test data divided by the total number of items. The test dataset has 200 items and the best algorithm, LightGbmMulti, scored 77.01 percent, which is 154 of 200 correct.

The MacroAccuracy is the average accuracy across the classes to predict. For example, suppose the 200-item test dataset had 60 conservative items, 90 moderate items and 50 liberal items. And suppose a model correctly predicted 45 of the 60 (0.7500) conservative items, 63 of the 90 (0.7000) moderate items, and 30 of the 50 (0.6000) liberal items. Then the MacroAccuracy for the model is (0.7500 + 0.7000 + 0.6000) / 3 = 0.6833.

MacroAccuracy is useful when a dataset is highly skewed toward one class. For example, if the test dataset had 180 conservative items, 10 moderate items and 10 liberal items, a model could just predict conservative for all items and score 180 / 200 = 0.9000 for MicroAccuracy, but the MacroAccurcy would be just (0.9000 + 0.0000 + 0.0000) / 3 = 0.3000. So, a big discrepancy between MicroAccuracy and MacroAccuracy values should be investigated.

## Using the Generated Model

Creating a machine learning prediction model is interesting, but the whole point is to use the model to make predictions. AutoML creates a subdirectory named SampleMulticlassClassification in the root People directory. You can specify a more descriptive name using the --name argument. The subdirectory contains directories that hold the generated model in .zip format and an auto-generated C# console application that can be used as a template for making predictions.

The auto-generated code is clear and easy to interpret. I double-clicked on file SampleMulticlassClassification.sln, which launched Visual Studio 2017, and then used the Solution file to load the C# project. The sample code makes a prediction using the first data item in the test dataset file. I edited the template code to make a prediction of the political leaning for a new, previously unseen person, as shown in **Figure 2**.

The edited prediction code begins by loading the trained model into memory and using the model to create a PredictionEngine object:

```
static void Main(string[] args)
{
  MLContext mlContext = new MLContext();
  ITransformer mlModel =
    mlContext.Model.Load(GetAbsolutePath(MODEL_FILEPATH),
    out DataViewSchema inputSchema);
  var predEngine =
    mlContext.Model.CreatePredictionEngine<ModelInput,
    ModelOutput>(mlModel);
...
```

The number of different combinations of algorithms and hyperparameters is unimaginably large.

Next, the custom code sets up the predictor values for a person:

```
Console.WriteLine("\nPredicting politic for Age = 33,
  Sex = Male, Region = central, Income = $62,000.00");
ModelInput X = new ModelInput();
X.Age = 33; X.Sex = false;
X.Region = "central"; X.Income = 62000.00f;
...
```

Recall that binary predictor variables such as Sex are Boolean-encoded. And notice that the Income variable has a trailing "f" to cast the value to type float, which is the default floating-point type used by ML.NET systems. The Age variable is also type float, but doesn't require a trailing "f" because the value doesn't contain a decimal point and is cast to type float automatically.

The prediction is made like so:

```
ModelOutput Y = predEngine.Predict(X);
string predPolitic = Y.Prediction;
float[] predProbs = Y.Score;
...
```

The Prediction property is a string representation of the predicted class ("moderate" for the demo) and the Score is an array of

float values that correspond to each possible class: (0.0034, 0.9055, 0.0912). The AutoML system uses the order in which class labels are first seen in the training data. Recall the training data looks like:

```
sex   age  region   income    politic
False  26  eastern  53800.00  conservative
False  19  western  39200.00  moderate
True   19  central  80800.00  liberal
False  52  eastern  86700.00  conservative
```

So "conservative" is [0], "moderate" is [1], and "liberal" is [2]. When I use AutoML I often rearrange the first few lines of my training data to get a nice order for the values to predict.

## Binary Classification and Regression

Once you understand the principles for using AutoML to create and use a prediction model for a multiclass classification problem, it's relatively simple to work with binary classification and regression problems. For example, you could issue the following command to create a model to predict a person's sex based on age, region, income and political leaning:

```
mlnet auto-train ^
--task binary-classification ^
--dataset ".\Data\people_train.tsv" ^
--test-dataset ".\Data\people_test.tsv" ^
--label-column-name sex ^
--max-exploration-time 300
```

And you could use this command to predict annual income based on age, sex and political leaning, but not region:

```
mlnet auto-train ^
--task regression ^
--dataset ".\Data\people_train.tsv" ^
--test-dataset ".\Data\people_test.tsv" ^
--ignore-columns region ^
--label-column-name income ^
--max-exploration-time 300
```

Binary classification and regression commands produce different result metrics than multiclass classification. Binary classification displays Accuracy, AUC, AUPRC and F1-score metrics. Briefly, AUC is "area under curve" of the receiver operating characteristics function and is a measure of how well two classes can be separated in a binary classification problem. Larger values of AUC are better. AUPRC is "area under the precision recall curve," which is a somewhat similar metric where larger values are also better. The F1 score is an average of precision and recall, both of which are metrics where larger values are better.

Using AutoML for regression displays R-squared, Absolute-loss, Squared-loss and RMS-loss. Larger values of R-square are better, and smaller values for Absolute-loss, Squared-loss and RMS-loss are better. If you're a relative newcomer to machine learning, don't get overly concerned with all these statistics. It's common practice in machine learning to give you many metrics. Accuracy is usually the primary metric to pay attention to.

But notice that there's no accuracy metric for a regression model. This is because there's no inherent definition of what a correct prediction is for a regression problem. For example, if a predicted annual income is $58,001.00 and the true annual income is $58,000.00, is the prediction correct?

For regression problems you must define a problem-dependent meaning of accuracy. Typically, you specify an allowable percentage difference. For example, if you specify a percentage of 0.10 and a correct income is $60,000.00, then any predicted income between $54,000.00 and $66,000.00 will be considered a correct prediction.

The template code generated by AutoML makes it easy for you to compute accuracy for a prediction problem. In pseudo-code:

```
loop each line in test dataset file
  parse out sex, age, region, politic, and correct income
  use sex, age, region, politic to compute predicted income
  if predicted is within x% of correct
    increment number correct
  else
    increment number wrong
end-loop
return number correct / (number correct + number wrong)
```

These examples should give you a good idea of the types of problems that can be tackled by AutoML and ML.NET. One type of machine learning that AutoML and ML.NET don't handle is prediction based on a neural network. Neural networks are significantly more complex than the traditional machine learning algorithms supported by AutoML. There has been discussion of adding neural network functionality to ML.NET and AutoML, but such functionality isn't likely to be added in the short term.

An interesting benefit of using AutoML is that, in addition to generating template code to load a trained model and use it to make predictions, AutoML generates a ModelBuilder.cs file that contains the underlying code that was used to create, train and save the prediction model. For example, some of the code generated in the ModelBuilder.cs file for the multiclass classification example is:

```
// Set the training algorithm
var trainer =
  mlContext.MulticlassClassification.Trainers.
  LightGbm(labelColumnName: "politic",
  featureColumnName: "Features").Append(mlContext.Transforms.Conversion.
  MapKeyToValue("PredictedLabel", "PredictedLabel"));

var trainingPipeline = dataProcessPipeline.Append(trainer);
```

If you want to explore creating prediction models using ML.NET manually in Visual Studio instead of automatically using AutoML, you can use the ModelBuilder.cs code as a starting point. This is much, much easier than writing the code from scratch.

## Wrapping Up

I'm quite impressed by my first look at AutoML. I could describe its cool technical features, but more important is that AutoML "just feels right." The CLI is simple and easy to use and the generated template code is clean and easy to modify. Put another way, AutoML feels like it's helping me instead of fighting me.

AutoML is just one part of a rapidly growing ecosystem of machine learning tools and systems. This ecosystem is expanding so quickly that even my colleagues and I, who work very close to the sources of these new systems, are finding it challenging to stay on top of everything. Systems that have been around for a couple of years, such as Azure Cognitive Services, Azure Machine Learning Studio, and Azure Data Science Virtual Machines, are being joined by newcomers such as Azure Data Prep SDK, NimbusML, and the ONNX Runtime. It's an exciting time to develop machine learning systems using .NET and open source technologies. ■

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

**Thanks** to the following Microsoft technical experts who reviewed this article: *Chris Lee, Ricky Loynd*

# Does your company create enterprise computing solutions for customers?

If so, does your ability to successfully win clients depend on their understanding of the vastly complex Microsoft stack?

**CHALLENGE**

### The Microsoft ecosystem is moving at a blistering pace

Your customers and prospects need clear guidance on some of the key architecture and directional questions for how to proceed.

**SOLUTION**

### Redmond Intelligence Research Reports

Actionable research and intelligence reports from our network of Microsoft experts, including Most Valuable Professionals (MVPs).

Sponsor a Redmond Intelligence Research Report, a Best Practices guide or a Solution Spotlight report. You'll come away with an authoritative, independent report, professionally edited and designed by the team behind Converge360, which you exclusively distribute. Contact us today for more details.

Redmond INTELLIGENCE
**redmondintelligence.com**

**CONTACT**

**Dan LaBianca** | General Manager | **Voice** 818.674.3416 / **E-mail** dlabianca@Converge360.com

CONVERGE 360
An 1105 MEDIA company

Visual Studio LIVE!
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

Sept 29 - Oct 3, 2019 | Westin Gaslamp Quarter
EXPERIENCE TECH TRAINING IN THE WARM CALIFORNIA SUN


# 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! San Diego, including everything Tuesday – Thursday at the conference.

**Save $400! When You Register by July 26**

*Your Adventure Starts Here!*

SUPPORTED BY
Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY
CONVERGE 360 An 1105 MEDIA company

vslive.com/sandiego

# AGENDA AT-A-GLANCE

#VSLive

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

| START TIME | END TIME | | | | |
|---|---|---|---|---|---|
| | | **Pre-Conference Full Day Hands-On Labs: Sunday, September 29, 2019** *(Separate entry fee required)* | | | |
| 7:30 AM | 9:00 AM | Pre-Conference Hands-On Lab Registration - Coffee and Morning Pastries | | | |
| 9:00 AM | 6:00 PM | **HOL01** Full Day Hands-On Lab: Develop an ASP.NET Core 2 App with EF Core 2 in a Day - *Philip Japikse* | | **HOL02** Day Hands-On Lab: Building a Modern DevOps Pipeline on Microsoft Azure with ASP.NET Core and Azure DevOps - *Brian Randell & Mickey Gousset* | |
| **START TIME** | **END TIME** | **Pre-Conference Workshops: Monday, September 30, 2019** *(Separate entry fee required)* | | | |
| 7:30 AM | 8:00 AM | Pre-Conference Workshop Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 5:00 PM | **M01** Workshop: Modern Security Architecture for ASP.NET Core - *Brock Allen* | **M02** Workshop: Kubernetes on Azure - *Vishwas Lele* | **M03** Workshop: Cross-Platform C# Using .NET Core, Kubernetes, and WebAssembly - *Rockford Lhotka & Jason Bock* | |
| 6:45 PM | 9:00 PM | Dine-A-Round | | | |
| **START TIME** | **END TIME** | **Day 1: Tuesday, October 1, 2019** | | | |
| 7:00 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **T01** MVVM and ASP.NET Core Razor Pages - *Ben Hoelting* | **T02** Azure, Windows and Xamarin: Using the Cloud to Power Your Cross-platform Applications - *Laurent Bugnion* | **T03** .NET Core and Azure, Microservices to Messaging - *Brady Gaster* | **T04** What's New in C# 7.X And C# 8 - *Philip Japikse* |
| 9:30 AM | 10:45 AM | **T05** Angular 101 - *Deborah Kurata* | **T06** Securing Web APIs from Mobile and Native Applications - *Brock Allen* | **T07** SQL Server 2019 Deep Dive - *Scott Klein* | **T08** How Microsoft Does DevOps - *Mickey Gousset* |
| 11:00 AM | 12:00 PM | **Keynote:** AI for the Rest of Us - *Damian Brady, Cloud Developer Advocate, Microsoft* | | | |
| 12:00 PM | 1:00 PM | Lunch | | | |
| 1:00 PM | 1:30 PM | Dessert Break - Visit Exhibitors | | | |
| 1:30 PM | 2:45 PM | **T09** Securing Single Page Applications (SPAs) - *Ben Hoelting* | **T10** To Be Announced | **T11** Data Pipelines with End-2-End Azure Analytics - *Scott Klein* | **T12** .NET Standard, .NET Core, why and how? - *Laurent Bugnion* |
| 3:00 PM | 4:15 PM | **T13** Managing Your Angular Async Data Effectively - *Deborah Kurata* | **T14** Cross-Platform Development with Xamarin, C#, and CSLA .NET - *Rockford Lhotka* | **T15** What Every Developer Ought to Know About #deeplearning and #neuralnetwork - *Vishwas Lele* | **T16** Azure Pipelines - *Brian Randell* |
| 4:15 PM | 5:30 PM | Welcome Reception | | | |
| **START TIME** | **END TIME** | **Day 2: Wednesday, October 2, 2019** | | | |
| 7:30 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **W01** Diving Deep Into ASP.NET Core 2.x - *Philip Japikse* | **W02** Building End-to-End ML Solution Using Azure Machine Learning Service - *Raj Krishnan* | **W03** Go Serverless with Azure Functions - *Eric Boyd* | **W04** Exceptional Development: Dealing With Exceptions in .NET - *Jason Bock* |
| 9:30 AM | 10:45 AM | **W05** WebAssembly: the Browser is your OS - *Jeremy Likness* | **W06** Azure Data Explorer—An in-depth Look at the New Microsoft PaaS Offering - *Raj Krishnan* | **W07** Keep Secrets with Azure Key Vault - *Eric D. Boyd* | **W08** Testability in .NET - *Jason Bock* |
| 11:00 AM | 12:00 PM | **General Session:** Moving .NET Beyond Windows = James Montemagno, Principal Program Manager - *Mobile Developer Tools, Microsoft* | | | |
| 12:00 PM | 1:00 PM | Birds-of-a-Feather Lunch | | | |
| 1:00 PM | 1:30 PM | Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win) | | | |
| 1:30 PM | 1:50 PM | **W09** Fast Focus: Getting Started with ASP.NET Core 2.0 Razor Pages - *Walt Ritscher* | **W10** Fast Focus: The Four Flavors of Power BI - *Thomas LeBlanc* | **W11** Fast Focus: Serverless of Azure 101 - *Eric D. Boyd* | |
| 2:00 PM | 2:20 PM | **W12** Fast Focus: Sharing C# Code Across Platforms - *Rockford Lhotka* | **W13** Fast Focus: Running .NET on Linux: What's Different? - *Steve Roberts* | **W14** Fast Focus: Lessons about DevOps from 3D Printing - *Colin Dembovsky* | |
| 2:30 PM | 3:45 PM | **W15** Angular vs React - a Comparison - *Gregor Dzierzon* | **W16** Effective Visualizations with Power BI - *Thomas LeBlanc* | **W17** Serverless .NET on AWS - *Steve Roberts* | **W18** Better Azure DevOps - Security 101 - *Brian Randell* |
| 4:00 PM | 5:15 PM | **W19** Object Oriented Programming Using TypeScript - *Gregor Dzierzon* | **W20** Transition from SSIS to Azure Data Factory - *Thomas LeBlanc* | **W21** Cloud Debugging – A Revolutionary Approach - *Alon Fliess* | **W22** DevOps for Machine Learning - *Damian Brady* |
| 7:00 PM | 8:30 PM | VSLive!'s City Lights at Night Trolley Tour | | | |
| **START TIME** | **END TIME** | **Day 3: Thursday, October 3, 2019** | | | |
| 7:30 AM | 8:00 AM | Registration - Coffee and Morning Pastries | | | |
| 8:00 AM | 9:15 AM | **TH01** Advanced Serverless Workflows with Durable Functions - *Jeremy Likness* | **TH02** Stunning Mobile Apps with the Xamarin Visual Design System - *James Montemagno* | **TH03** Building a Stronger Team, One Strength at a Time - *Angela Dugan* | **TH04** Creating Business Applications Using Blazor (Razor Components) - *Michael Washington* |
| 9:30 AM | 10:45 AM | **TH05** What's New in Bootstrap 4 - *Paul Sheriff* | **TH06** How to Have Better Business Intelligence through Visualizations - *Walt Ritscher* | **TH07** How Do You Measure Up? Collect the Right Metrics for the Right Reasons - *Angela Dugan* | **TH08** Building Business Applications Using Bots - *Michael Washington* |
| 11:00 AM | 12:15 PM | **TH09** Signal R: Real-time for All Things - *Brady Gaster* | **TH10** Microsoft Power Platform, RAD Done Right: Building Dynamic Mobile Apps with PowerApps - *Walt Ritscher* | **TH11** Past, Present & Future of C# Debugging - *Alon Fliess* | **TH12** Automate Your Life with PowerShell on Lambda - *Steve Roberts* |
| 12:15 PM | 1:30 PM | Lunch | | | |
| 1:30 PM | 2:45 PM | **TH13** Advanced Fiddler Techniques - *Robert Boedigheimer* | **TH14** Building UWP Apps for Multiple Devices - *Tony Champion* | **TH15** To Microservice or Not to Microservice? How? - *Alon Fliess* | **TH16** Testing in Production Using Azure and Visual Studio Team Services (VSTS) - *Colin Dembovsky* |
| 3:00 PM | 4:15 PM | **TH17** Improving Web Performance - *Robert Boedigheimer* | **TH18** Building Cross Device Experiences with Project Rome - *Tony Champion* | **TH19** Getting Started with Unit Testing in Visual Studio - *Paul Sheriff* | **TH20** Modernizing Your Source Control: Migrating to Git from Team Foundation Version Control (TFVC) - *Colin Dembovsky* |

*Speakers and sessions subject to change*

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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



DINO ESPOSITO

# ASP.NET Core gRPC Services

Originally developed at Google, gRPC today is a remote procedure call (RPC) framework that has emerged as an alternative to RESTful and HTTP-based interfaces to connect remote components and specifically microservices. The new RPC framework was created in part to work with modern technologies such as HTTP/2 and Protobuf.

The gRPC framework offers native bindings for a number of programming languages, including C#, and using it from within ASP.NET and ASP.NET Core microservices has never been an issue. It's worth mentioning, though, that earlier .NET implementations of gRPC wrapped native DLLs and kept the service hosted on its own server. In .NET Core 3.0, however, a gRPC service is a full .NET implementation hosted on Kestrel, like other ASP.NET Web applications. This article explores the Visual Studio 2019 project template and the dedicated ASP.NET Core layer.

## Creating a gRPC Service in Visual Studio 2019

When you choose to create a new ASP.NET Core Web application, Visual Studio 2019 gives you the opportunity to create a new kind of component—a gRPC service. If you go ahead and complete the wizard, you end up with a minimal ASP.NET Core project with the canonical pair of files, startup.cs and program.cs, plus a couple of unusual new folders named protos and services. The program.cs file is nothing special, but the startup.cs file is worth a look.

When you choose to create a new ASP.NET Core Web application, Visual Studio 2019 gives you the opportunity to create a new kind of component—a gRPC service.

The Configure method of the Startup class contains the following line:

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

As expected, AddGrpc is an extension method of the IServiceCollection class. If you're a curious person and want to dig into the internals of the method, here's a summary of what you'll find (probably not information you'll ever use, but it's like having a look under the hood!):

```
services.AddRouting();
services.AddOptions();
services.TryAddSingleton<GrpcMarkerService>();
services.TryAddSingleton<ServiceMethodsRegistry>();
services.TryAddSingleton(typeof(ServerCallHandlerFactory<>));
```

The call to AddRouting is a bit more functional as it serves to enable the use of global routing (a new feature introduced with ASP.NET Core 2.2) for communication between gRPC clients and the gRPC endpoints in the ASP.NET Core service being built. The three singletons added to the service runtime environment take care of the overall management of the service lifecycle—factory, discovery and invocation.

In the Configure method of the application's startup class, you'll find declared the use of the global routing system and the necessary endpoints, as follows:

```
app.UseRouting();
app.UseEndpoints(endpoints =>
{
  endpoints.MapGrpcService<GreeterService>();
});
```

A gRPC service isn't based on controllers, rather it uses a service class that's constructed for every request to process the client call. This service class is called GreeterService in the sample code that the template generates. The MapGrpcService<T> method creates a binding of the URL of the gRPC call to a call handler that gets invoked when processing the request. The handler factory is retrieved and used to create a new instance of the T class for the requested action to take place. This is a key difference between the ASP.NET Core 3.0 implementation of gRPC and the existing C# implementation. Note, however, that it's still possible that an instance of the service class is resolved as a singleton from the DI container.

## The Prototype of the Service Class

In the sample template-based project, the service class is defined as follows:

```
public class GreeterService : Greeter.GreeterBase
{
  ...
}
```

At first, you may think that implementing a gRPC service is, much like plain controller classes, all about writing a class with a bunch of public action methods. Well, not exactly. Sure, the Visual Studio project contains a file with a class defined as shown in the previous code. However, there's no place in the same

project where the base class of the service class in the snippet—the Greeter.GreeterBase class—is defined. How is that possible? The answer lies in the source code of another file—a small text file—placed in the protos folder (see **Figure 1**).

The protos folder contains one or more text files with the .proto extension, known as Protocol Buffer definition files or Protobuf files for short. Usually, there's one for each service found in the ASP.NET Core service, but there are no actual constraints. In fact, you can define a service in one file and the messages it uses in another file, or define multiple services in the same file. The .proto text file provides the interface definition of the service. Here's the content of the .proto file for the sample greeter service:

```
syntax = "proto3";
package Greet;
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

If the first piece of content isn't a syntax element, the proto compiler will assume an older proto2 syntax. The newest .proto syntax is proto3 instead. Next, you find the name of the package to be created.

The service section indicates the application programming interface, namely the list of callable endpoints. Note that SayHello is a unary method, namely a method that works as a normal function call—the client sends a single request to the server and gets a single response back. Also, gRPC supports client, server and bi-directional streaming methods. These enable the client to write messages to a stream and get one response, or send one message and get a stream of return messages, or share a stream with a server to read and write. You describe streaming methods in a Protobuf file using the keyword stream, like so:

```
rpc SayHello1 (HelloRequest) returns (stream HelloReply) {}
rpc SayHello2 (stream HelloRequest) returns (HelloReply) {}
rpc SayHello3 (stream HelloRequest) returns (stream HelloReply) {}
```

The message sections in the proto file refer to message types being used by the methods. In other words, the message section defines any of the custom data-transfer object types being used by the public methods. In the sample snippet, the service named Greeter is made of one RPC method called SayHello. The method takes an input parameter of type HelloRequest and returns an output value in the form of an instance of the type HelloReply. Both types are made by a single string property. The integer value (in this case, 1) assigned to the properties of both messages indicates the field number and determines the position of the specific content in the message binary format being transferred across the wire. Hence, those values shouldn't be changed once the service is deployed.

## Details of Message Elements

In the definition of the message types in the proto file, you can use a number of primitive types including bool, int32, sint32, double, float and a long list of other numeric variations (int64, fixed32, uint32 and more). You can also use the bytes type for any arbitrary sequence of bytes. In particular, the sint32 type is ideal for signed integers as this format results in a more efficient encoding of negative values. Note that the aforementioned types are those defined in the proto3 syntax. Each supported language (such as C#) will then turn them into language-specific types. In C#, you can have long, bool, int, float and double. Properties are given a default value that coincides (at least in C#) with the default value of the language type.

The schema of a message is variable to some extent, in the sense that any declared property is optional. If marked with the repeated keyword, however, the same property is still optional, but can be repeated more than once. This explains why the field number is relevant, as it's used as the placeholder for the actual content, as shown here:

```
message ListOfCitiesResponse {
  repeated string city = 1;
}
```

In addition to primitive scalar types, the proto syntax also supports enums. Enum types can be defined in the proto file or even inline in the body of the message type, like so:

```
enum Department {
  Unknown = 0;
  ICT = 1;
  HR = 2,
  Accounting = 3;
}
```

Note that the first element is required to take the value of 0. You're allowed to have members with the same value as long as you

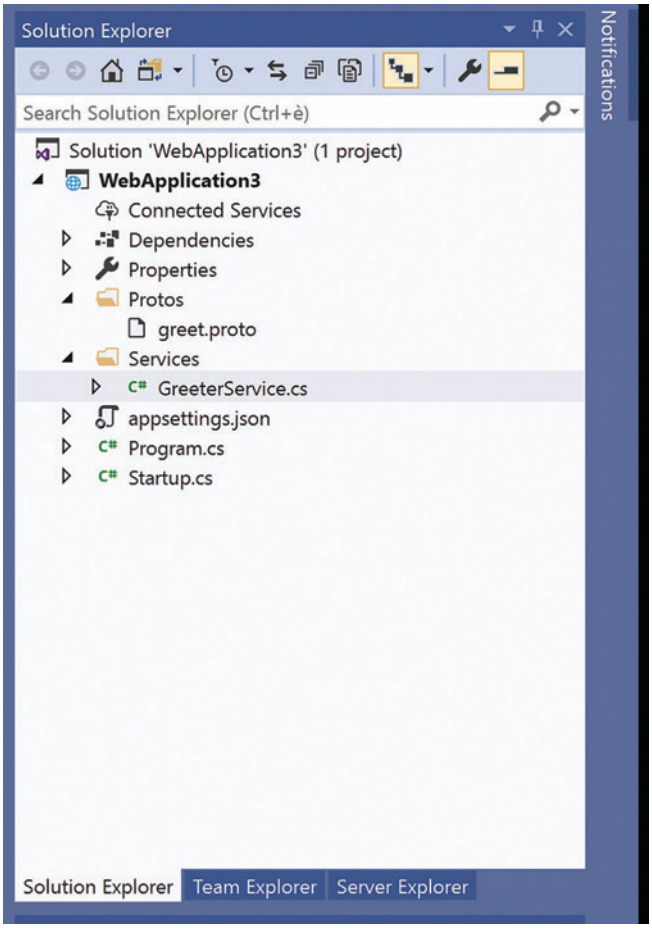



Figure 1 **Solution Folder of a gRPC Project**

Figure 2 **Embedded Definition of Enum Types**

```
message EmployeeResponse {
  string firstName = 1;
  string lastName = 2;
  enum Department {
    Unknown = 0;
    ICT = 1;
    HR = 2;
  }
  Department currentDepartment = 3;
}
message ContactResponse {
  ...
  EmployeeResponse.Department department = 3;
}
```

Figure 3 **Auto-Generated rRPC Service Class**

```
public static partial class Greeter
{
  public abstract partial class GreeterBase
  {
    public virtual Task<Greet.HelloReply> SayHello(
      Greet.HelloRequest request,
      ServerCallContext context)
    {
      throw new RpcException();
    }
    ...
  }
}
```

declare it through the allow_alias option, as shown here:

```
enum Department {
  option allow_alias = true;
  Unknown = 0;
  ICT = 1;
  HR = 2,
  Accounting = 3;
  Finance = 3;
}
```

Without the allow_alias option, you'll get a compile error in case of repeated enum values. If the enum is defined globally in the proto file, you can just use it by name in the various message types. If it's defined in the body of a message type, you can still prefix its name with the name of the message type. **Figure 2** shows this.

You can freely reference all message types in the same proto file, and also those defined in external proto files, as long as you import them first. Here's the code:

```
import "protos/another.proto";
```

Properties of a message type aren't limited to scalar types and enums. A message element can refer another message type, whether its definition is embedded in the message, global in the proto file, or imported from another proto file.

If during the lifetime of the service you need to update one of the message types, just pay attention not to reuse the field number. You can always add new elements, as long as the code knows how to deal with clients that may send packets devoid of the extra element. You're also welcome to remove fields. In this case, though, it's crucial that the number of the removed field isn't reused. This, in fact, may cause confusion and conflicts. To protect against this, you can declare the critical field number as reserved, like so:

```
message PersoneResponse {
  reserved 1, 2, 5 to 8;
  reserved "gender", "address";
  ...
}
```

You can reserve field numbers (also using the extended syntax N to M), as well as field names. The same applies to entries in an enum type. This said, however, most of the time you're better off just renaming the field with some prefix like NOTUSED_.

This explanation doesn't exhaust all the possible variations of the proto3 syntax. For more information, refer to bit.ly/2Hz5NJW.

## The Actual Service Class

The source code of the .proto file is silently processed to generate a base class—the missing Greeter.GreeterBase class—that provides the plumbing for the gRPC client/server communication to take place. You'll find the actual source code of the base in the \Debug\netcoreapp3.0 folder of the project. **Figure 3** shows an excerpt.

This file is generated after you run build and won't exist before then. Also note that your project must have a reference to the Grpc.Tools package.

Aside from the .proto text file and the under-the-hood work to compile it to a base class, the resulting service class isn't really different from a plain MVC controller. As you can see, the class is made of a few overridden public methods and their actual implementation:

```
public override Task<HelloReply> SayHello(
  HelloRequest request, ServerCallContext context)
{
  return Task.FromResult(new HelloReply
  {
    Message = "Hello " + request.Name
  });
}
```

In the body of the method, you can do whatever makes sense for the specific task, calling a database or external service, or performing any due calculation. For the service class to receive calls,

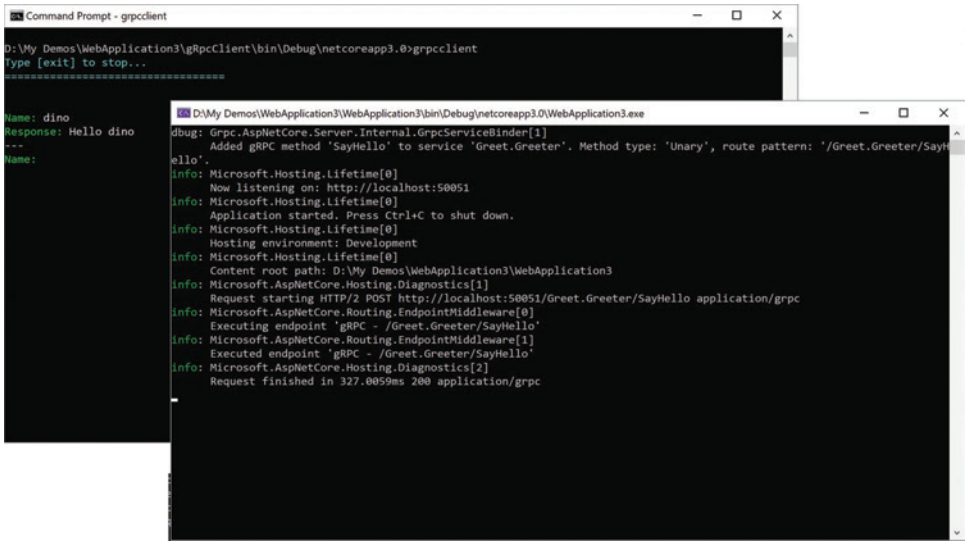



Figure 4 **Client and Server Applications in Action**

you must start the gRPC server to listen over a configured port. Let's have a look at a gRPC client now.

## Writing a gRPC Client

The ASP.NET Core server application has dependencies on the ASP.NET Core gRPC package, as well as the core Google.ProtoBuf protocol. It also has a dependency on the Grpc.Tools package, but not for runtime action. The package is responsible for processing the content of the proto text file. A client application can be a console application with dependencies on the Grpc.Core package alone, the Google.ProtoBuf package and the Grpc.Tools.

To add a dependency on the actual service, you have two options. One is adding a reference to the proto file and let the tooling do the job. The other is creating a third (class library) project that only contains the proto file. Next, you link the resulting assembly to both a client and a server project. To reference the proto file, you copy the protos folder from the server project in the client project and edit the CSPROJ file, adding the following code:

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />
</ItemGroup>
```

Next, you write the code to open a channel and place calls. It's worth noting that the gRPC framework uses the ProtoBuf binary protocol for remote calls, which in turn uses HTTP/2 as the transport. (Note that ProtoBuf is the default setting, but theoretically you can use other serialization/deserialization stacks.) Here's the necessary setup for calling a gRPC service. Note that by the time ASP.NET Core 3.0 ships, there will be a managed gRPC client that will change the previous code for building a client:

```
var channel = new Channel(serviceUrl, ChannelCredentials.Insecure);
var client = new Greeter.GreeterClient(channel);
```

By default, the service URL is localhost and the port (as configured in the server project) is 50051. From the client reference you call prototyped methods as if it were a local call, as shown in the following code:

```
var request = new HelloRequest { Name = name };
var response = client.SayHelloAsync(request);
Console.WriteLine(response.Message);
```

You can see the resulting output in **Figure 4**.

In the end, gRPC has a noticeable analogy with old-fashioned Distributed COM of the 1990s. Like DCOM, it allows you to call remote objects as if they were local, and it does this over a binary and super-fast protocol leveraging HTTP/2.

gRPC isn't REST and it's not perfect, but it's another option and fully open-sourced. It's probably premature to say whether gRPC will replace REST in the heart of developers. For sure, some concrete and realistic microservices scenarios already exist where gRPC is really beneficial. You can find a useful comparison between gRPC and REST here: bit.ly/30VB7do. ■

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

**Thanks** to the following Microsoft technical experts for reviewing this article: *John Luo, James Newton-King*

Visual Studio LIVE! EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS | SQL Server LIVE! TRAINING FOR DBAs AND IT PROS | TECHMENTOR IN-DEPTH TRAINING FOR IT PROS | Office & SharePoint LIVE! ON-PREMISE, CLOUD, & CROSS-PLATFORM TRAINING | Artificial Intelligence LIVE! AI FOR DEVELOPERS AND DATA SCIENTISTS | Cloud & Containers LIVE! CLOUD-NATIVE, PAAS & SERVERLESS COMPUTING

# The Ultimate Education Destination

## LIVE! 360

TECH EVENTS WITH PERSPECTIVE

## 6 Great Events, 1 Low Price!

**November 17-22, 2019 | ORLANDO**
Royal Pacific Resort at Universal


**Live! 360** brings the IT and developer community together for a unique conference, featuring 6 co-located conferences for (almost) every IT title. Customize your learning by choosing from hundreds of sessions, dozens of track topics, workshops and hands-on labs from hundreds of expert speakers.


EVENT PARTNER Microsoft

SILVER SPONSOR aws

SUPPORTED BY msdn magazine | Redmond Channel Partner | Redmond MAGAZINE | VIRTUALIZATION & Cloud Review | Visual Studio MAGAZINE

PRODUCED BY CONVERGE 360 An 1105 MEDIA company

## Visual Studio LIVE!

EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

Visual Studio Live! features unbiased and practical developer training on the Microsoft Platform. Explore hot topics such as ASP.NET Core, JavaScript, Xamarin, Database Analytics and more!

## SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

Whether you are a DBA, developer, IT Pro or Analyst, SQL Server Live! will provide you with the skills you need to drive your data to succeed.

## Office & SharePoint LIVE!

ON-PREMISE, CLOUD, & CROSS-PLATFORM TRAINING

Office & SharePoint Live! provides leading-edge training to administrators and developers who must customize, deploy and maintain SharePoint Server on-premises and in Office 365.

## TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

TechMentor gives a strong emphasis on doing more with the tech you already own plus solid coverage of what is next - striking the perfect balance of training essentials for the everyday IT Pro.

## Artificial Intelligence LIVE!

AI FOR DEVELOPERS AND DATA SCIENTISTS

Artificial Intelligence Live! offers real-world training on the languages, libraries, APIs, tools and cloud services you need to implement real AI and Machine Learning solutions today and into the future.

## NEW! IN 2019 Cloud & Containers LIVE!

CLOUD-NATIVE, PAAS & SERVERLESS COMPUTING

Cloud & Containers Live! covers both IT infrastructure and software development aspects of cloud-native software design, development, release, deployment, operations, instrumentation/monitoring and maintenance.

## Who Should Attend:

- Developers
- IT Professionals
- Database Administrators
- Business Intelligence Professionals
- SharePoint Professionals
- and more!


**Save $500 when you register by August 16!**

Promo Code MSDN

**live360events.com/orlando**



# Where Are They Now?

I've been writing this column for almost a decade, a very long time in this business. Along the way, I've visited with and written about some pretty remarkable people and events. So at the risk of waxing nostalgic, here's a retrospective of some people and things I've written about and how they're doing today.

Gísli Ólafsson (November 2014) somehow managed to avoid catching Ebola during his trips to Liberia at the height of the 2013-2016 outbreak. He continues his humanitarian work, flying to wherever a natural disaster hits, most recently Mozambique after Cyclone Itai. Check out his TEDx Talk at youtu.be/ittc7_OoaYc, and his book, "The Crisis Leader," at amzn.to/2KbkvbQ.

Hugh Blair-Smith (May 2016) is enjoying his retirement, waxing contemplative at the 50th anniversary of the moon landing, which his computer enabled. He was good for a quote when I first met him, and he's good for one now. Says Blair-Smith: "I enjoy Buzz Aldrin's T-shirts emblazoned 'Get Your Ass to Mars' but there aren't a lot of good suggestions as to what one's ass should do when it gets there. The only thought I find compelling is setting up a permanent settlement so that the 100 best people on Earth could relocate there just before we finish screwing our home planet. Hope that's enough to avoid inbreeding—want to be on the selection committee?"

*MSDN Magazine*'s April 2011 guest columnist Simba died that August, aged 20 years and two months. We have Starlight and Toothless now, and they're great. But they're not Simba.

Amazon's Mayday button (May 2014), which I loved on my Fire tablet, unfortunately was removed in June of 2018. Whenever I had any trouble, I'd just tap Mayday. A live, English-speaking person would appear on my screen, walk me through whatever problem I had, drawing on my screen to show me where to tap. I can still get help by calling Amazon on the phone, but that's much less convenient. The live agents must have cost too much. Too bad Alexa can't do that job yet.

The winners of Microsoft's Imagine Cup programming contest (October 2011) have gone from good (Barcelona, 2003) to excellent (New York, 2011) to [expletive] amazing (Seattle, 2019). Bryan Chiang, an 18-year-old UCLA freshman, figured out how to deduce blood glucose level by examining a snapshot of a patient's eye, taken with a low-cost adapter on an ordinary smartphone. The process uses convolutional neural networks that Chiang developed with Azure Virtual Machines (see bit.ly/2W77jqA). Think of the millions of daily needle sticks no longer needed to measure a diabetic patient's sugar level, the billions of dollars no longer spent on testing meters and supplies; all replaced by a selfie (September 2015) and some cogitation.

*MSDN Magazine*'s April 2011 guest columnist Simba died that August, aged 20 years and two months. We have Starlight and Toothless now, and they're great. But they're not Simba.

Annabelle's FIRST robotics team continues to compete and amaze. As their mentor, I've watched them fall down, learn, get up, fall again, learn some more; always getting up one more time than they fell down. Our ranking in the standings wasn't great, but we won the District Engineering Inspiration Award, the second-highest award that FIRST bestows, for "outstanding success in advancing respect and appreciation for engineering within a team's school or organization and community." That's a fitting acknowledgment for the blood, sweat, tears and soldering iron burns these students offered up. They blasted Queen's song, "We Are the Champions" as we drove home, singing along at the tops of their lungs. And, indeed, they are champions, regardless of their finishing place. If you've never done one of these gigs, I can't explain it to you. And if you have, then I don't need to.

Which brings me to daughter Annabelle, guest author of my September 2014 and September 2017 columns, graduating high school as I write these words. Through hard work and personal drive, she won admission to her first choice—Olin College of Engineering, in Needham, Mass.—and I won a stack of college bills. (How about some consulting help or in-house classes this fall, dear reader?)

Annabelle will be packing up as you read these words. And I'm left to recall the scene from "Fiddler on the Roof," where Hodel takes leave of Tevye to follow her beloved Perchik on his exile to Siberia, saying: "Papa, God alone knows when we shall see each other again." To which Tevye replies: "Then we will leave it in His hands." As the train pulls away and the smoke fades into the endless Russian steppe, Tevye looks heavenward and says: "Take care of her. See she dresses warm." Amen. ■

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

# Address the ELEPHANT IN THE ROOM

## Bad address data costs you money, customers and insight.

Melissa's 30+ years of domain experience in address management, patented fuzzy matching and multi-sourced reference datasets power the global data quality tools you need to keep addresses clean, correct and current. The result? Trusted information that improves customer communication, fraud prevention, predictive analytics, and the bottom line.


- Global Address Verification
- Digital Identity Verification
- Email & Phone Verification
- Location Intelligence
- Single Customer View

---

**+ .NET**
**+ Microsoft® SSIS**
**+ Microsoft® Dynamics CRM**

See the Elephant in Your Business -

**Name it and Tame it!**

melissa

www.Melissa.com | 1-800-MELISSA

**Free API Trials, Data Quality Audit & Professional Services.**