THE MICROSOFT JOURNAL FOR DEVELOPERS

JUNE 2018 VOL 33 NO 6

msdn magazine

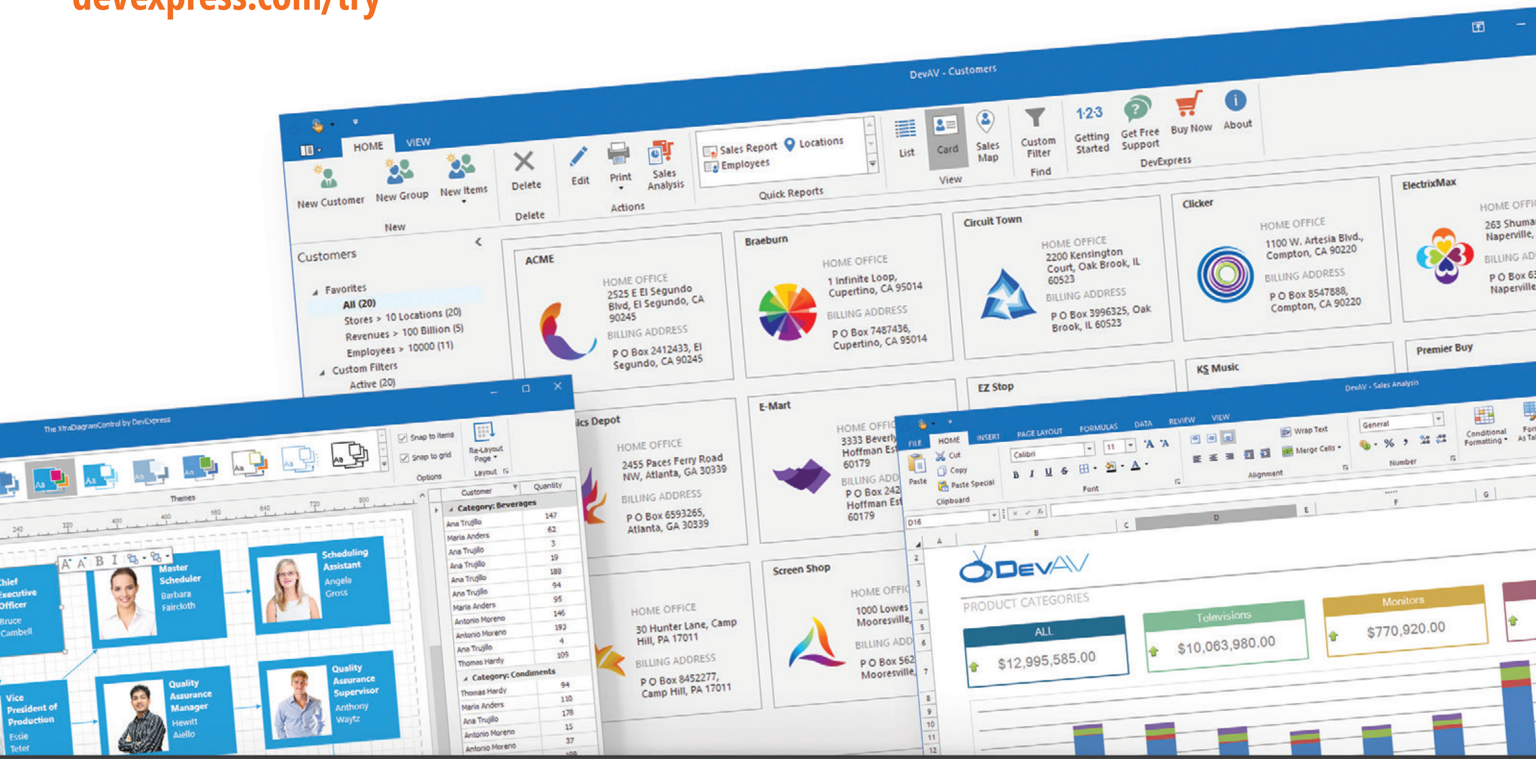
C#

DevExpress®

The King of Desktop Development

Your peers have voted DevExpress Desktop Components best-in-class for 6 straight years. We invite you to see why. Download your free 30-day trial today.

devexpress.com/try



Visual Studio MAGAZINE READERS CHOICE 2017 2016 2015 2013 2012

All trademarks or registered trademarks are property of their respective owners.

DevExpress®

Your Next Great Desktop App Starts Here

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

Symbol		Last Price	Volume	Volume Dynamics
MSFT	⬇	24.07	115458922	
EBAY	⬇	17.28	13928481	
ERTS	⬇	20.72	7715607	
YHOO	⬇	15.8	20326040	
CSCO	⬆	18.92	71703553	
AAPL	⬇	139.48	25780518	
IBM	⬆	105.89	13562225	
MSI	⬆	25.02	5806050	
ORCL	⬇	20.66	51092929	
AMZN	⬇	82.96	5441504	
INTC	⬇	16.01	71510905	
SNE	⬇	26.23	795334	

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

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

JUNE 2018 VOL 33 NO 6

msdn magazine

C#

Microsoft

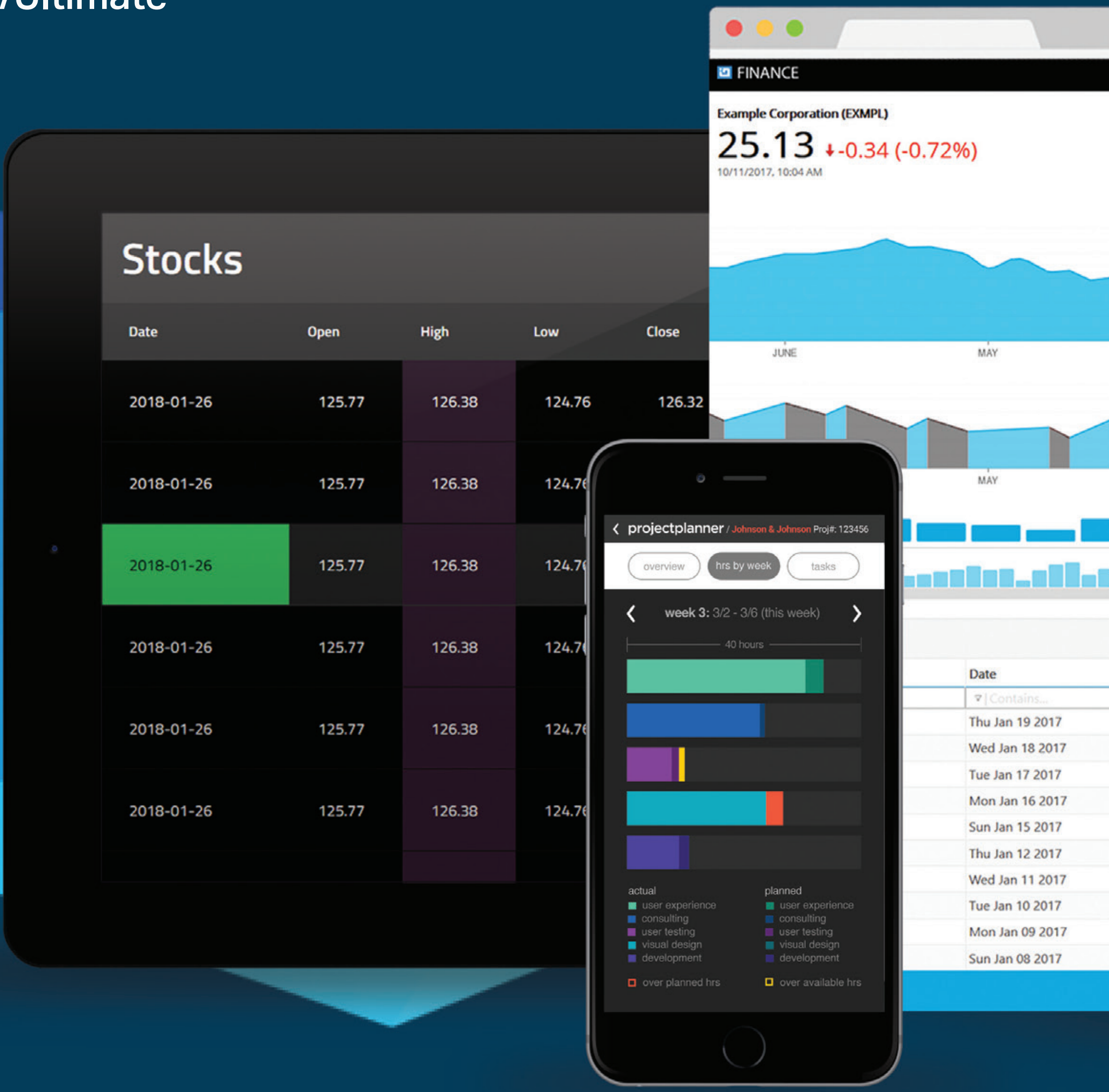
INFRAGISTICS

Faster Paths to Amazing Experiences

Infragistics Ultimate includes 100+ beautifully styled, high performance grids, charts & other UI controls, plus productivity tools for building web, desktop and mobile apps.

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

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



Stocks

Date	Open	High	Low	Close
2018-01-26	125.77	126.38	124.76	126.32
2018-01-26	125.77	126.38	124.76	
2018-01-26	125.77	126.38	124.7	
2018-01-26	125.77	126.38	124.7	
2018-01-26	125.77	126.38	124.76	
2018-01-26	125.77	126.38	124.76	

FINANCE

Example Corporation (EXMPL)

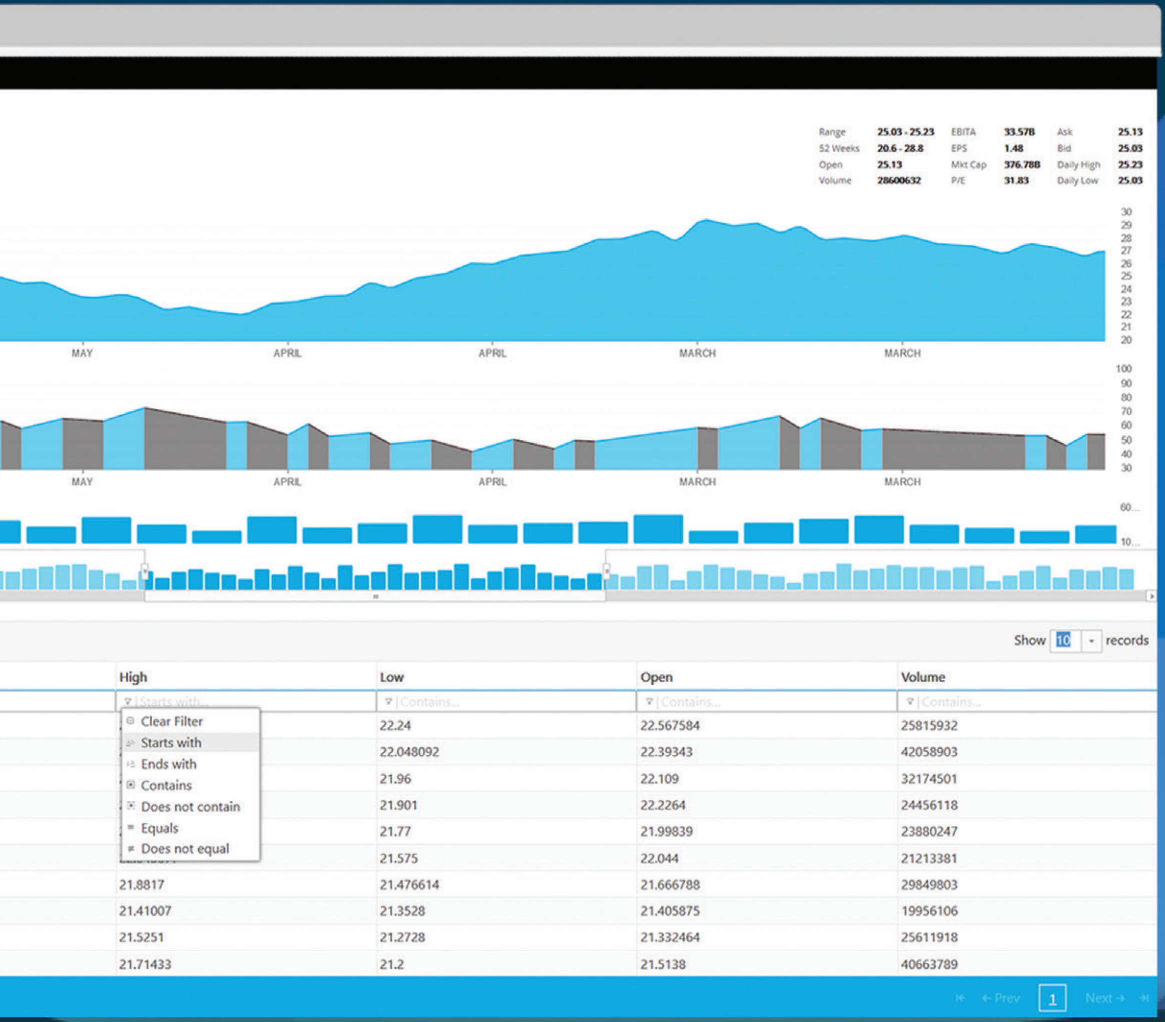
25.13 ↓-0.34 (-0.72%)

10/11/2017, 10:04 AM

Date
Contains...
Thu Jan 19 2017
Wed Jan 18 2017
Tue Jan 17 2017
Mon Jan 16 2017
Sun Jan 15 2017
Thu Jan 12 2017
Wed Jan 11 2017
Tue Jan 10 2017
Mon Jan 09 2017
Sun Jan 08 2017

New Release
Infragistics Ultimate 18.1

- ✓ Fastest **grids** & **charts** on the market for the Angular developer
- ✓ The most complete **Microsoft Excel & Spreadsheet Solution** for .NET & JavaScript
- ✓ UI controls designed to meet the demands of the toughest **finacial & capital market apps**



Range	25.03 - 25.23	EBITA	33.57B	Ask	25.13
52 Weeks	20.6 - 28.8	EPS	1.48	Bid	25.03
Open	25.13	Mkt Cap	376.78B	Daily High	25.23
Volume	28600632	P/E	31.83	Daily Low	25.03

High	Low	Open	Volume
[illegible]	22.24	22.567584	25815932
[illegible]	22.048092	22.39343	42058903
[illegible]	21.96	22.109	32174501
[illegible]	21.901	22.2264	24456118
[illegible]	21.77	21.99839	23880247
[illegible]	21.575	22.044	21213381
21.8817	21.476614	21.666788	29849803
21.41007	21.3528	21.405875	19956106
21.5251	21.2728	21.332464	25611918
21.71433	21.2	21.5138	40663789

msdn magazine

JUNE 2018 VOLUME 33 NUMBER 6

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

ENTERPRISE COMPUTING GROUP

President
Henry Allain

Chief Revenue Officer
Dan LaBianca

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

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

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

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

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

ENTERPRISE COMPUTING GROUP EVENTS
Vice President, Events **Brent Sutton**
Senior Director, Operations **Sara Ross**
Senior Director, Event Marketing **Mallory Bastionell**
Senior Manager, Events **Danielle Potts**
Senior Marketing Coordinator, Events **Michelle Cheng**

1105 MEDIA INC
YOUR GROWTH. OUR BUSINESS.

Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Chief Financial Officer
Craig Rucker

Chief Technology Officer
Erik A. Lindgren

Executive Vice President
Michael J. Valenti

Chairman of the Board
Jeffrey S. Klein

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

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o *MSDN Magazine*, 4 Venture, Suite 150, Irvine, CA 92618.

LEGAL DISCLAIMER The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

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

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

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

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

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

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA A Century Moving Business Forward

WPA Western Publications Association

LEADTOOLS®

One SDK, Multiple Platforms

LEADTOOLS toolkits will help you create desktop, web, server, and mobile applications with the greatest collection of programmer-friendly and cross-platform document, medical, and multimedia technologies.

OCR FORMS BARCODE DICOM PACS + MANY MORE

Microsoft .NET iOS macOS

Get Started Today
DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

Our Intelligent Future

In my Editor's Note column, "Groundhog Day," in the February issue (msdn.com/magazine/mt829266), I honored the contributions of weather prognosticating rodents everywhere, with a collection of industry predictions from *MSDN Magazine* columnists. Those predictions ranged from the serious (Ted Neward expecting more "side-channel" attacks along the lines of Spectre and Meltdown), to the silly (David Platt waiting for a university to replace its science department with StackOverflow).

A common theme did emerge, though, and that was around artificial intelligence (AI) and its impact on software development. Frank La Vigne, author of the Artificially Intelligent column, said that within two years AI would become a "mandatory skill set" for mainstream developers, much the way Web and later mobile development have. "It sounds far-fetched," La Vigne told me at the time, "but then so did the idea of Web development being a mainstream enterprise development platform during the height of the client/server era."

Microsoft is working to converge and comingle its AI and cloud efforts, and opening some amazing application scenarios in the process.

Test Run author James McCaffrey concurred, predicting that sophisticated tooling would drive adoption and spur enterprise developers to add prediction code to their responsibilities, much the way they took on design, implementation and test.

The Microsoft Build 2018 Conference, held May 7-10 in Seattle, has proven both authors right. At the show, Microsoft Chief Executive Officer Satya Nadella and Executive Vice President Scott Guthrie articulated the company's expanding AI and machine learning (ML) strategy, debuting powerful new tools and platform capabilities that span the gamut from the cloud to Internet of Things (IoT) devices at the edge. The new technologies, Microsoft noted, are designed "to help every developer be an AI developer."

If you've been too busy to pay it much attention, now might be a good time to carve out some time to put a little AI into your life, and to get familiar with tools like Visual Studio Tools for AI, Microsoft Cognitive Services and Azure Machine Learning Workbench.

There's good reason to make the investment. Microsoft is working to converge and comingle its AI and cloud efforts, and opening some amazing application scenarios in the process. For instance, Microsoft is enabling the family of Cognitive Services APIs (starting with Custom Vision) for the Azure IoT Edge runtime, making it possible for an edge device, like a camera-equipped drone and field truck, to execute intelligent decision making without cloud connectivity. Efforts like Project Kinect for Azure, the Project Brainwave neural net processing architecture, and improved Cognitive Services like Custom Vision and unified Speech Service all point to Microsoft's strategic effort to improve, extend and enrich the AI development landscape.

We're also seeing Microsoft apply AI directly to the developer experience. At Build, the company previewed Visual Studio IntelliCode, which leverages AI to provide intelligent suggestions that can boost code quality and productivity.

Frank La Vigne says Microsoft's efforts in the AI space play directly to what it does best: "Whether it was bringing the GUI to the masses in the '80s or developer tools to coders in the '90s, Microsoft has a long history of democratizing technology. In fact, I'd argue that is the company's core strength."

La Vigne says this year's Build event represents an important moment for Microsoft, as it works to introduce rank-and-file developers to a new frontier in development.

"I see the world changing. In fact, it already has—it's just that not everyone in the mainstream has realized it yet," La Vigne, says. "What I'm interested to see is the reaction of mainstream enterprise developers to the announcements and general tone at Build."

[illegible]

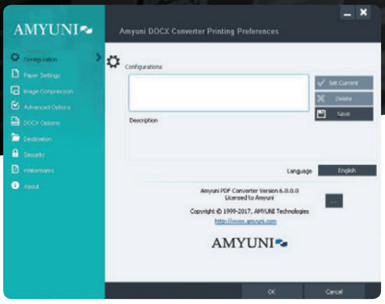
Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

DOCXCONVERTER
For Windows

Free Demo at DOCXConverter.com

Amyuni DOCX Converter for Windows

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



Amyuni PDF Converter Version 6.0.0.0
Licensed to Amyuni
Copyright © 1999-2017, AMYUNI Technologies
http://www.amyuni.com

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

Create

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

Convert

Convert images and graphics of multiple formats into DOCX shapes

OCR

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

Extract

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

Open

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

Configure

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

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

Powered by Amyuni Technologies:

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

www.docxconverter.com

JULIE LERMAN

Replacing a Bulky API with Azure Functions

Back in 2015 and 2016, I wrote a few columns about a Node.js Web API that interacted with Azure DocumentDB. There was a lot involved. Even though I took advantage of the Node.js SDK designed for DocumentDB, I still had to write a lot of logic. There was logic for building up objects that represented the DocumentDB account, the database and the collections. There was code to create queries and to execute them—layers of asynchronous calls. But I was able to write a nice little API that allowed my application to create, query and update data in the database.

In more recent columns, I've worked with Azure Functions, Microsoft's serverless APIs that live in Azure and integrate seamlessly with other Azure technologies. One of those integrations is with Azure Cosmos DB, the data storage into which Azure DocumentDB evolved.

Having written a number of Azure Functions, integrating them with Cosmos DB to read and write data, I looked back at my older Node.js APIs and realized I could eliminate about 98 percent of the code by converting them into Azure Functions. That's because most of the code I had written was to interact with the database, but now Azure Functions take care of all of that work. With nothing more than a configuration to identify the connection string—whether the data is coming from the database or going into it—and the relevant query, the built-in features of the function will take care of the rest.

Using Visual Studio Code and Extensions

In the earlier Azure Functions columns, I worked directly in the Azure portal. But it's also possible to use the Azure Functions Core Tools to develop on your computer and then deploy to Azure. The first version of the tools (version 1) runs only on Windows, with .NET 4.7.1 or higher, and there's a Visual Studio 2017 extension for working with those tools.

In contrast, version 2 of the Azure Functions Core Tools runs on .NET Core and is cross-platform, and a Visual Studio Code extension lets you work with it handily (`bit.ly/2H7VmxH`). That's the version I'll be working with in this article. Note that both the tools and the extension are in preview.

Working with this extension has a few setup requirements, which you'll find in the Read Me for the Visual Studio Code extension. And it assumes you've already installed Visual Studio Code and its prerequisites on your system, which could be Windows, macOS or Linux.

The extension helps you easily create new Azure Function App folders to contain Azure Functions, create the functions (with a variety of templates to choose from), run and debug them locally, and deploy them to Azure. I've been really impressed because it takes away the hard work and all you have to do is focus on your code—which is the promise of serverless computing.

In addition to the Azure Functions extension, I also installed the Azure Cosmos DB extension for Visual Studio Code (`bit.ly/2HkPfDE`). I'm using that to look at the collections and documents of my existing databases, but you can also use the extension to create new Cosmos DB accounts, databases and collections, as well as retrieve and update documents and create new documents. I've put in a suggestion to be able to import documents given pre-existing JSON files and hope to see that in there someday.

I've been really impressed because it takes away all the hard work and all you have to do is focus on your code—which is the promise of serverless computing.

Critical to using both of these extensions is the ability to sign into your Azure account from Visual Studio Code, so I've also installed the Azure Account extension (`bit.ly/2k1phdp`).

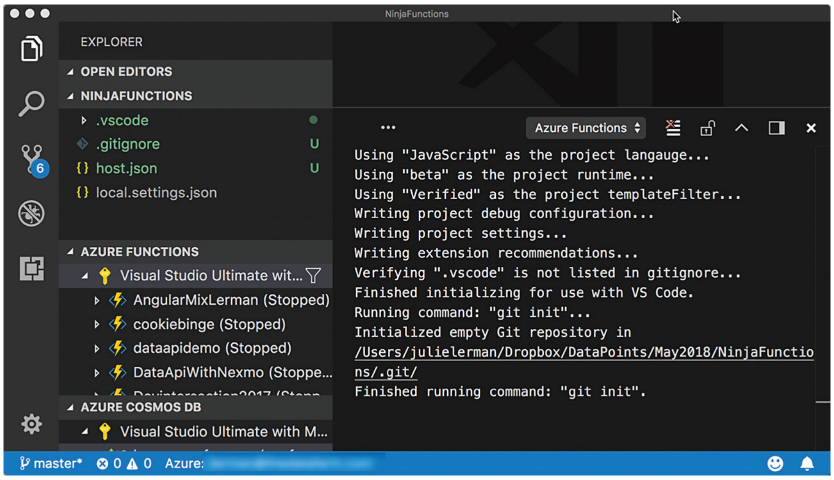
Each of these three extensions have handy walk-throughs on the pages I've linked to, which will help you get started using them. I'll do a light walk-through here for the purpose of showing you how much easier it is to create the Azure Function counterparts of my Node.js Web API than it was to build all of the code and dependencies I relied on for the Node.js solution. I will, however, still use JavaScript as the language for my functions. JavaScript and Node.js support is built into Visual Studio Code.

Creating the Azure Functions Project

After installing the three extensions and their dependencies, it was time to recreate my API. I started by creating a new folder on my

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

Code download available at msdn.com/magazine/0618magcode.



NinjaFunctions

EXPLORER

OPEN EDITORS

NINJAFUNCTIONS

- .vscode
- .gitignore U
- host.json U
- local.settings.json

AZURE FUNCTIONS

- Visual Studio Ultimate wit...
  - AngularMixLerman (Stopped)
  - cookiebinge (Stopped)
  - dataapidemo (Stopped)
  - DataApiWithNexmo (Stoppe...

AZURE COSMOS DB

- Visual Studio Ultimate with M...

```
Using "JavaScript" as the project langauge...
Using "beta" as the project runtime...
Using "Verified" as the project templateFilter...
Writing project debug configuration...
Writing project settings...
Writing extension recommendations...
Verifying ".vscode" is not listed in gitignore...
Finished initializing for use with VS Code.
Running command: "git init"...
Initialized empty Git repository in
/Users/julielerman/Dropbox/DataPoints/May2018/NinjaFunctio
ns/.git/
Finished running command: "git init".
```

master* ⊗ 0 ⚠ 0 Azure: [redacted]

Figure 1 **Visual Studio Code After Creating the New Azure Functions Project**

computer, named NinjaFunctions, then opening it in Visual Studio Code. If you're following along, begin by using the Azure Functions extension to turn the folder into an Azure Functions folder. You can do this by clicking the Create New Project icon (which looks like a folder), on the toolbar of the Azure Functions pane. Then follow the prompts to select the already opened NinjaFunctions folder and the language. I'm choosing JavaScript. The extension will then add some files needed for the functions into your folder, and initialize a Git repository. **Figure 1** shows the IDE with the results of this process in the Azure Functions output window, as well as the files (.vscode, .gitignore, host.json and local.settings.json) it added. You can also see the Azure Functions pane listing all of my cloud-based Azure Function Apps and, below that, my Cosmos DB databases, thanks to that extension.

The goal of the first function is to read an existing document supplied in JSON format and add it to the new collection.

Creating an Azure Function in the Project

With the project set up, it's time to create the first function. This function will import JSON files into my Cosmos DB database. Note that I've already created the new database account (lermandatapoints) but I haven't yet created a database or a collection. I'll do that using the Azure Cosmos DB extension. In the extension pane, right-click on the Cosmos DB account where you want to create the database and choose Create Database from the menu. Type in the new name when prompted (mine, naturally, is Ninjas) and this database will be created in your Azure account using default settings. Feel free to tweak those settings in the portal if necessary. Finally, create a collection in the database by right-clicking the new database, choosing Create Collection and providing its name. I'll be boring and call my collection Ninjas, as well. You can leave the partition key blank for this little demo.

The goal of the first function is to read an existing document supplied in JSON format and add it to the new collection. If you've read my recent Azure Functions articles in this column, you may recall that one is called "Creating Azure Functions That Can Read from Cosmos DB with Almost No Code" (msdn.com/magazine/mt829268). In this case, I'll be creating a function that can *write* to Cosmos DB with almost no code.

I won't be building a front end. Instead, I'll be using the Postman application to construct calls to my APIs and pass the JSON document in as a message body. You can do the same with Fiddler, as well.

Back on the Azure Functions extension pane, click the Create Function icon to create a new function in the project folder. I want this function to respond to HTTP requests, so select HTTP trigger and then provide a name. Mine is AddNinjaDocument. Following the prompts, make the function anonymous so it's easier to test without having to provide credentials.

In response to this action, the extension creates a new folder with three files:

- function.json, which contains the default configuration for the function
- index.js file for the logic
- a sample data file

Configuring the Function

Like the function created in my January 2018 column, "Creating Azure Functions to Interact with Cosmos DB" (msdn.com/magazine/mt814991), this function will output data to a Cosmos DB database. In the earlier article, I used the portal to configure this integration and the portal wrote my choices into a function.json file. This time, I'll define the configuration manually in function.json. Open the file and you'll see two integrations already defined, based on the fact that the HTTP trigger template was used. After the second integration, add a comma and then copy in this cosmosDB configuration, which specifies the name of the document to send to the database, the type of integration (cosmosDB), the database and collection names, the connection setting, and the direction of the integration (out to the database):

```
{
  "name": "outputDocument",
  "type": "cosmosDB",
  "databaseName": "Ninjas",
  "collectionName": "Ninjas",
  "createIfNotExists": true,
  "connectionStringSetting": "mydbconnection",
  "direction": "out"
}
```

I can define the connection string in the local.settings.json file so I don't have to hard code it into the function.json file. Local.settings.json is the local representation of the app.settings.json file that will live in the portal and contain your application secrets.

So, here, I'm just saying that the connection string can be found in a setting called mydbconnection.

The setting goes in the Values section of local. settings.json. You can copy the connection string by right-clicking the Cosmos DB account in the Cosmos DB extension pane and then pasting it into the json file. The string will start with:

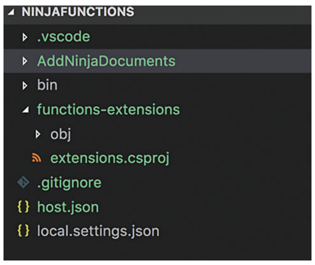
```
"mydbconnection":
  "AccountEndpoint=https://yourdb.documents.azure.com/
```

In an upcoming enhancement to the Azure Function Core Tools, the presence of the cosmosDB type in the function.json file will trigger the extension to automatically install a package with the logic needed to run and debug your project locally. At this time, though, you'll need to install this package manually. Let's do that now.

In the terminal window, be sure you're pointed to the root folder, NinjaFunctions, and enter the following Azure Functions CLI command:

```
func extensions install -p Microsoft.Azure.WebJobs.Extensions.CosmosDb -v 3.0.0-beta7
```

Note that I'm installing the beta version that's current as I'm writing this article. You can check its NuGet page for the latest version if you do have to install it manually (bit.ly/2EyLNCw). This will create a new folder called functions-extensions in your project (see **Figure 2**). The folder contains a .NET Standard 2.0 project, which is driven by the project file, extensions.csproj. The csproj code listing is in **Figure 3**.



```
NINJAFUNCTIONS
  .vscode
  AddNinjaDocuments
  bin
  functions-extensions
    obj
    extensions.csproj
  .gitignore
  host.json
  local.settings.json
```

Figure 2 **The New Functions-Extensions Folder to House the Cosmos DB Extension**

Adding the Tiny Bit of Code

Let's get back to the AddNinjaDocuments function. The function.json file is all set, so the next step is to complete the index.js file, which is where the function's logic lives. You can delete all of the default sample code. All you really need for the function is code to bind the incoming JSON to the function's output; in other words, what gets sent to the Cosmos DB collection:

```
module.exports = function (context, req) {
  context.bindings.outputDocument=req.body;
  context.done();
};
```

But it's helpful to have a little bit of debugging info. So, rather than the pure minimalist version, replace the entire contents of the default sample code with the following:

```
module.exports = function (context, req) {
  context.log('HTTP request received.');
  try {
    context.bindings.outputDocument = req.body;
  }
  catch (error) {
    context.log(error);
  }
  context.bindings.res = { status: 201, body: "Insert succeeded." };
  context.done();
};
```

All you really need for the function is code to bind the incoming JSON to the function's output; in other words, what gets sent to the Cosmos DB collection.

The JavaScript code is quite different from the C# function code I wrote in the previous articles. You can learn about the structure of JavaScript in Azure Functions at bit.ly/2GQ9eJt.

The context being passed in to the function is used by the runtime to pass data in and out of the function. I named the Cosmos DB binding "outputDocument" in the function.json configuration. Now I'm setting that binding to whatever body is encapsulated in the incoming HTTP request. The template created a binding named res for the HTTP response, which I'm using to relay the success of the function. The context.done method signals to the runtime that the function is complete.

Running the Function in Visual Studio Code

So, that's all there is to the function! Everything else is taken care of by the application settings, the function configuration and Azure Functions APIs. You can go ahead and run the function in Visual Studio Code. While it's absolutely possible to debug, set breakpoints and explore variables, let's just run the function app, which you can do in the terminal with the command:

```
func start
```

This will display the brightly colored Azure Functions logo in the terminal window and then output some processing info. At the end of all this output, you'll see the URL where the function is running. If there are multiple functions in your project, the URLs will be listed separately for each. In Postman or Fiddler, build a

Figure 3 **The extensions.csproj Folder with References to the Cosmos DB Extension**

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
      <WarningsAsErrors></WarningsAsErrors>
      <DefaultItemExcludes>**</DefaultItemExcludes>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.Azure.WebJobs.Extensions.CosmosDb"
                      Version="3.0.0-beta7" />
    <PackageReference
      Include="Microsoft.Azure.WebJobs.Script.ExtensionsMetadataGenerator"
      Version="1.0.0-beta2" />
  </ItemGroup>
</Project>
```

Figure 4 **JSON Code for a Document To Be Inserted into the Database**

```
{
    "Name": "Kacy Catanzaro",
    "ServedInOniwaban": false,
    "Clan": "American Ninja Warriors",
    "Equipment": [
      {
        "EquipmentName": "Muscles",
        "EquipmentType": "Tool"
      },
      {
        "EquipmentName": "Spunk",
        "EquipmentType": "Tool"
      }
    ],
    "DateOfBirth": "1/14/1990"
  }
```

DevExpress®

DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

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

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

Free 30-day trial

devexpress.com/spreadsheet

#UseTheBest

POST request using this URL. In the body section you can paste in the JSON listed in **Figure 4** and then send the request. **Figure 5** shows my Postman UI, with the request URL and body displayed, along with the response.

Verifying the Insert

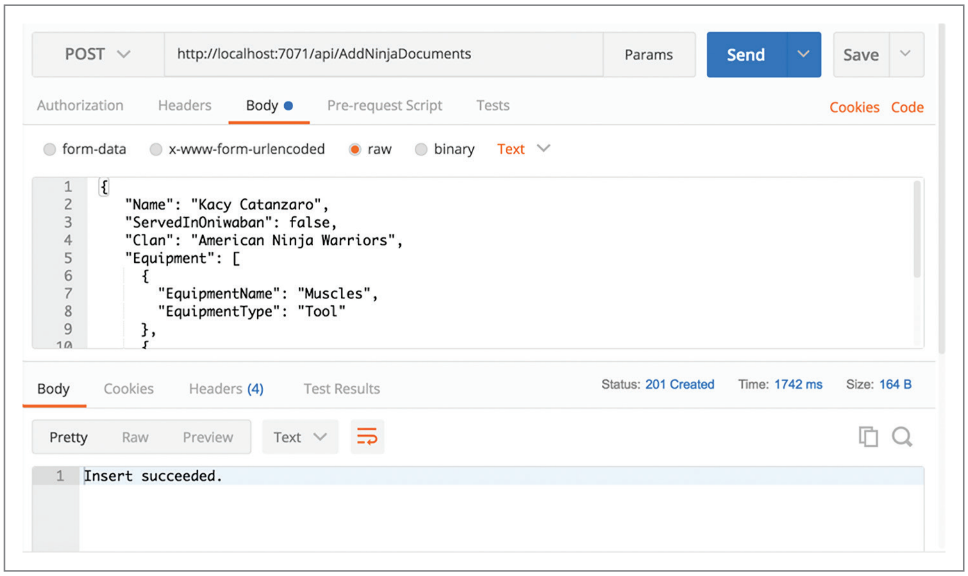
While the HTTP response in **Figure 5** does say that all went well, it's nice to actually see the data that was sent to the database in the cloud. And thanks to the Azure Cosmos DB extension, you can verify that directly in Visual Studio Code. First, let's be sure that the function app stops running. In the terminal window, press CTRL+C to shut down the host. You may need to press CTRL+C a second time to get the prompt back in the terminal.

Now, in the Cosmos DB extension, expand the account, the database and the collection. You may need to refresh the view with the refresh icon on the extension's pane. Within the collection, you can see the document that was just added. Select that to open it in the editor where you'll see not just the data you added, but the metadata added by Azure Cosmos DB, as shown in **Figure 6**.

Next Steps

The download that accompanies this article also includes two more functions I created to replace the other methods from the original Node.js API. One is to return data filtered by a name part and the other to return data based on the id. With these, rather than having output binding to Cosmos DB, there's an input binding that defines the database query using the SQL API. The HTTP response spits out the JSON representation of the query results.

The context being passed in to the function is used by the runtime to pass data in and out of the function.

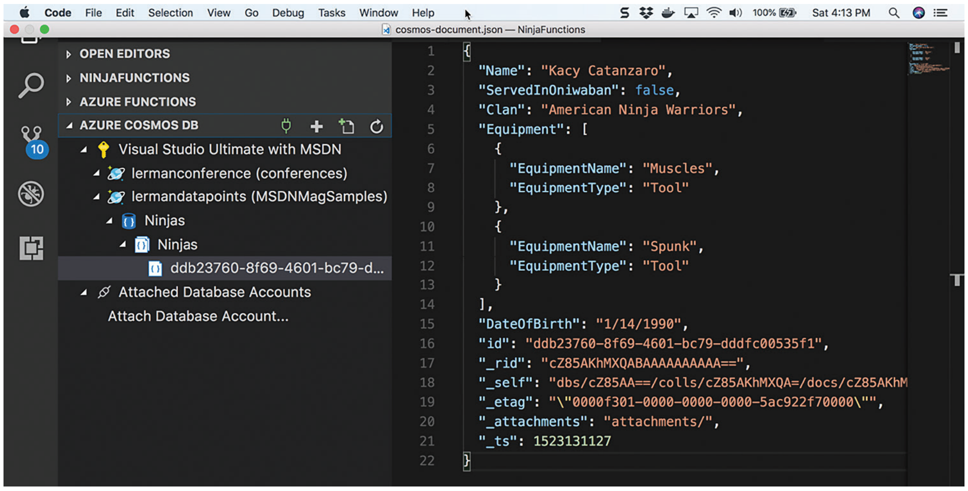


```
{
    "Name": "Kacy Catanzaro",
    "ServedInOniwaban": false,
    "Clan": "American Ninja Warriors",
    "Equipment": [
      {
        "EquipmentName": "Muscles",
        "EquipmentType": "Tool"
      },
      {
```

Status: 201 Created   Time: 1742 ms   Size: 164 B

```
Insert succeeded.
```

Figure 5 **Creating a Request to Call the AddNinjaDocuments Function**



Code File Edit Selection View Go Debug Tasks Window Help
100%
Sat 4:13 PM
cosmos-document.json — NinjaFunctions
OPEN EDITORS
NINJAFUNCTIONS
AZURE FUNCTIONS
AZURE COSMOS DB
Visual Studio Ultimate with MSDN
lermanconference (conferences)
lermandatapoints (MSDNMagSamples)
Ninjas
Ninjas
ddb23760-8f69-4601-bc79-d...
Attached Database Accounts
Attach Database Account...
10

```
{
  "Name": "Kacy Catanzaro",
  "ServedInOniwaban": false,
  "Clan": "American Ninja Warriors",
  "Equipment": [
    {
      "EquipmentName": "Muscles",
      "EquipmentType": "Tool"
    },
    {
      "EquipmentName": "Spunk",
      "EquipmentType": "Tool"
    }
  ],
  "DateOfBirth": "1/14/1990",
  "id": "ddb23760-8f69-4601-bc79-dddfc00535f1",
  "_rid": "cZ85AKhMXQABAAAAAAAAAA==",
  "_self": "dbs/cZ85AA==/colls/cZ85AKhMXQA=/docs/cZ85AKhM
  "_etag": "\"0000f301-0000-0000-0000-5ac922f70000\"",
  "_attachments": "attachments/",
  "_ts": 1523131127
}
```

Figure 6 **The New Document as Displayed by the Cosmos DB Extension**

Once you've tested and debugged your Azure Functions in Visual Studio Code, you can also use the extension to publish the functions to your Azure account. You can learn more about that from the extension's ReadMe.

I recommend taking a look at the download from my original article, which is also in a GitHub repository at github.com/julielerman/AureliaDocDB. The API is in the models folder and, as you'll see, there's a lot more code involved with making read and write calls into the DocumentDB than for the Azure Functions, thanks to its integration with Cosmos DB. I think in my coding future, I'll always consider Azure Functions as the first line of defense when it's time to write another Web API! ■

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

Thanks to the following Microsoft technical expert for reviewing this article: *Jeff Hollan*

How To Be MEAN: Reactive Programming

Welcome back again, MEANers.

In the previous two articles (msdn.com/magazine/mt829391 and msdn.com/magazine/mt814413), I talked a great deal about what Angular refers to as "template-driven forms," which are forms described by Angular templates and extended by the use of code to control validation and other logic. That's all well and good, but a new concept about how to view the "flow" of information—and the resultant control exhibited in code—within a Web application is starting to seize developers' imaginations.

I speak, of course, of "reactive" programming, and one of the great shifts between AngularJS (that is, Angular 1.0) and Angular (Angular 2.0 and beyond) is that Angular now has explicit support for a reactive style of programming, at least at the Angular level. All of which is to say, Angular supports using reactive styles of programming within Angular without having to commit to a completely reactive style of architecture outside of Angular.

Before I dive in, though, it helps to make sure everybody's on board with what "reactive" means here.

Reactive Reactions

Fundamentally, reactive styles of programming are easy to understand, so long as you're willing to completely invert your perspective on the traditional way of doing UI programming. (Yes, that's a joke. But only a small one.)

In the traditional MVC-based approach for building UIs, the UI elements (buttons, text fields and the like) are created, and then … you wait. The system processes messages from the hardware, and when some combination of messages indicates that the user did something—clicked a button, typed into a text box or whatever—the code bound to the controls fires. That code typically modifies the model lying behind the UI, and the application goes back into waiting.

This "event-driven" style of programming is intrinsically asynchronous, in that the code that the programmer writes is always driven by what the user does. In some ways, it would be better to call it "indeterministic" programming, because you can't ever know what the user will do next. Angular sought to make this easier by creating two-way binding between the UI and data model, but it can still be tricky to keep the order of things straight.

The reactive style, however, takes a more one-way approach. Controls are constructed in code (rather than in the template), and you programmatically subscribe to events emitted by those controls to respond to the user doing things. It's not quite React-style reactive programming, but it's pretty close, and it still permits the same kind of "immutable data model" style of programming that has enamored a percentage of the industry.

Reactive Construction

To begin, let's look at constructing the SpeakerUI component in a reactive manner, rather than in a template-driven manner. (Here, I'm going to go with the more traditional Angular convention and call it SpeakerDetail, but the name is largely irrelevant to the discussion.) First, in order to help simplify what I'm working with, I'll use the abbreviated form of Speaker and SpeakerService, as shown in **Figure 1**.

Notice that the SpeakerService is using the Promise.resolve method to return a Promise that is instantly resolved. This is an

Figure 1 **Speaker and SpeakerService**

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

export class Speaker {
  id = 0
  firstName = ""
  lastName = ""
  age = 0
}

@Injectable()
export class SpeakerService {
  private speakers : Speaker[] = [
    {
      id: 1,
      firstName: 'Brian',
      lastName: 'Randell',
      age: 47,
    },
    {
      id: 2,
      firstName: 'Ted',
      lastName: 'Neward',
      age: 46,
    },
    {
      id: 3,
      firstName: 'Rachel',
      lastName: 'Appel',
      age: 39,
    }
  ]

    getSpeakers() : Promise<Array<Speaker>> {
    return Promise.resolve(this.speakers);
  }

  getSpeaker(id: number) : Promise<Speaker> {
    return Promise.resolve(this.speakers[id - 1 ]);
  }
}
```

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS
#VSLIVE25

August 13 – 17, 2018
Redmond, WA
Microsoft Headquarters

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

Join Us at Microsoft HQ for Intense Developer Training

DEVELOPMENT TOPICS INCLUDE:

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

Help Us Celebrate 25 Years of Coding Innovation!

Visual Studio Live! (VSLive!™) is celebrating 25 years as one of the most respected, longest-standing, independent developer conferences, and we want you to be a part of it. **Join us from August 13 – 17** as developers, software architects, engineers, designers and more come together at Microsoft Headquarters for 5 days of unbiased education on the Microsoft Platform. Hone your skills in **Visual Studio, ASP.NET Core, AngularJS, SQL Server,** and so much more.

$400 Super Early Bird Savings Ends June 8!
Use Promo Code MSDN

REGISTER NOW

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/redmond

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS
#VSLIVE25

August 13 – 17, 2018
Redmond, WA
Microsoft Headquarters

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

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

JOIN US INSIDE THE TECH MECCA: MICROSOFT HQ!

Visual Studio Live! Redmond is back at Microsoft Headquarters this August, where you can experience the high-tech features of this state-of-the-art facility while you network with peers, code with experts and enjoy some additional unique perks with us in Redmond, including:

- More sessions from Microsoft insiders who are "in the trenches."
- Visiting the Microsoft Company Store and Visitor Center.

$400 Super Early Bird Savings Ends June 8!
Use Promo Code MSDN

REGISTER NOW

EVENT SPONSOR Microsoft
PLATINUM SPONSOR SMARTBEAR
SILVER SPONSOR Progress
SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE
PRODUCED BY 1105 MEDIA YOUR GROWTH. OUR BUSINESS.

vslive.com/redmond

Figure 2 **HTML Template**

```
<h2>Select Speaker</h2>
<form [formGroup]="selectGroup">
<label>Speaker:
  <select formControlName="selectSpeaker">
    <option *ngFor="let speaker of speakers"
            [value]="speaker.lastName">{{speaker.lastName}}</option>
  </select>
</label>
</form>

<h2>Speaker Detail</h2>
<form [formGroup]="speakerForm" novalidate>
  <div>
    <label>First Name:
      <input formControlName="firstName">
    </label>
    <label>Last Name:
      <input formControlName="lastName">
    </label>
    <label>Age:
      <input formControlName="age">
    </label>
  </div>
</form>
```

easy way to mock out the service without having to build a Promise object in the longer fashion using the Promise constructor.

Next, the SpeakerDetail component is just a standard Angular component ("ng new component speaker-detail"), and the constructor should inject a SpeakerService instance as a private constructor parameter. (This was detailed in my August 2017 column, "How To Be MEAN: Angular Plays Fetch" [msdn.com/magazine/mt826349].) While you're at it, use the constructor to call the SpeakerService's getSpeakers method to get back the array, and store that locally into the component as a property called "speakers." So far, this sounds a lot like the template-based component described earlier. The HTML template for this component will display a dropdown of all the speakers in the system (as obtained by getSpeakers), and then as each is selected, display the details in another set of controls underneath that dropdown. Thus, the template looks like **Figure 2**.

It may seem strange that the alternative to "template-based" forms uses HTML templates. That's largely because the reactive-forms approach doesn't do away with the template, but instead removes most of the logic away from the template and into the component. This is where things start to take a left turn from the techniques examined in the previous pair of columns. The component code will actually construct a tree of control objects, and most (if not all) interaction with the controls in these two forms will happen entirely inside the code base. I won't put event-handlers in the template. Instead, I'll hook them up inside the component code.

But first, I need to construct the controls themselves. These will all be FormControl instances, imported from the @angular/forms module, but it can get a little tedious to construct them each (along with any required validations) by hand in code. Fortunately, Angular provides a FormBuilder class that's designed to make things a bit more succinct and compact to construct a whole form's worth of controls, particularly for longer (or nested) forms.

In the case of my speaker form—where I want to have a dropdown serve as a selection mechanism to choose which speaker in the list to work on—I need to do something a little different. I want to have two forms: one around the dropdown speaker-selection control, and the other containing the detail for the individual speaker. (Normally this would be two separate components in a master-detail kind of arrangement, but I haven't covered routing yet.) Thus, I need two forms, and in **Figure 3** I show how to construct them, both with the FormBuilder and without.

Figure 3 **Constructing Forms**

```
export class SpeakerDetailComponent implements OnInit {
  selectGroup : FormGroup
  speakerForm : FormGroup
  speakers : Speaker[]

  constructor(private formBuilder : FormBuilder,
              private speakerService : SpeakerService) {
    speakerService.getSpeakers().then( (res) => { this.speakers = res; });
    this.createForm();
  }

  createForm() {
    this.selectGroup = new FormGroup({
      selectSpeaker : new FormControl()
    });

    this.speakerForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      age: [0]
    });
  }

  // ...
}
```

This is, of course, only an excerpt of the class—there's a few more things I need to add before it's ready to ship, but this code neatly demonstrates two of the ways to construct a form. The "selectGroup" is the FormGroup that contains the one FormControl for the HTML <select> control, and I use the SpeakerService to populate a local array of Speaker instances so the <select> can populate itself. (This is actually on the template, not in the component code. If there's a way to populate the dropdown from the component code, I've not found it yet in Angular 5.)

The second form, called speakerForm, is populated using the FormBuilder, which is a tiny little DSL-like language for constructing controls. Notice how I call "group" to indicate that I'm constructing a group of controls, which are effectively name-value pairs. The name is the name of the control (which must match the formControlName property in the template for each control) and the value is either an initial value, or an initial value followed by an array of validation functions (two arrays, if you want to include validation functions that run asynchronously) to run to validate the value users type in to the control.

This constructs the two forms, but selecting something in the dropdown doesn't do anything. I need to respond to the event the dropdown will broadcast, and I can do that by hooking a function on to the "valueChanges" method of FormControl, like so:

```
export class SpeakerDetailComponent implements OnInit {
  // ...
  ngOnInit() {
    const speakerSelect = this.selectGroup.get('selectSpeaker');
    speakerSelect.valueChanges.forEach(
      (value:string) => {
        const speaker = this.speakers.find( (s) => s.lastName === value);
        this.populate(speaker);
      }
    );
  }
  // ...
}
```

Notice that the valueChanges property is actually a stream of events. Thus, I use forEach to indicate that for each event that comes through, I want to take the value to which the control has changed (the parameter to the lambda) and use that to find the speaker by last name in the array returned by the SpeakerService. I then take that Speaker instance and pass it to the populate method, shown here:

```
export class SpeakerDetailComponent implements OnInit {
  // ...
  populate(speaker) {
    const firstName = this.speakerForm.get('firstName');
    const lastName = this.speakerForm.get('lastName');
    const age = this.speakerForm.get('age');

    firstName.setValue(speaker.firstName);
    lastName.setValue(speaker.lastName);
    age.setValue(speaker.age);
  }
  // ...
}
```

The populate method simply grabs the reference to the FormControls from the speakerForm group, and uses the setValue method to populate the appropriate value from the Speaker to the appropriate control.

The developer retains full control over what happens when.

Reactive Analysis

A fair question to ask at this point is how this is any better than the template-based forms shown earlier. Frankly, it's not a question of better—it's a question of different. (The Angular docs even say this outright in the official documentation page on Reactive Forms at angular.io/guide/reactive-forms.) Capturing most of the logic in the component code (rather than in the template) may strike some developers as being more logical or cohesive, but reactive forms also have the advantage that they're essentially a reactive "loop": Because the controls don't directly map to the underlying model object. The developer retains full control over what happens when, which can simplify a number of scenarios immensely.

The Angular team doesn't believe that one approach to forms is inherently any better than the other, in fact specifically stating that both styles can be used within the same application. (I wouldn't try to use both within the same component, however, unless you understand Angular really deeply.) The key here is that Angular supports two different styles of gathering input from users, and both are valid and viable for a good number of scenarios. But what happens if you have a large number of different forms that need to be displayed, or the form that needs to be displayed depends on the underlying model object, which can change? Angular offers yet another option, which you'll see in the next column. Happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor, currently working as the director of Engineering and Developer Relations at Smartsheet.com. He has written a ton of articles, authored or 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 Microsoft technical expert: *Garvice Eakins*

Tuple Trouble: Why C# Tuples Get to Break the Guidelines

Mark Michaelis

Back in the August 2017 issue of *MSDN Magazine* I wrote an in-depth article on C# 7.0 and its support for tuples (`msdn.com/magazine/mt493248`). At the time I glossed over the fact that the tuple type introduced with C# 7.0 (internally of type ValueTuple<…>) breaks several guidelines of a well-structured value type, namely:

- ***Do Not*** declare fields that are public or protected (instead encapsulate with a property).
- ***Do Not*** define mutable value types.
- ***Do Not*** create value types larger than 16 bytes in size.

These guidelines have been in place since C# 1.0, and yet here in C# 7.0, they've been thrown to the wind to define the System.ValueTuple<…> data type. Technically, System.ValueTuple<…> is a family of data types of the same name, but of varying arity (specifically, the number of type parameters). What's so special about this particular data type that these long-respected guidelines no longer apply? And how can our understanding of the circumstances in which these guidelines apply—or don't apply—help us refine their application to defining value types?

This article discusses:

- Guidelines for programming value types (structs)
- How to use C# 7.0 tuples to override Equals and GetHashCode
- Understanding Guidelines are just tha—they still allow for coloring outside the lines given sufficient justification

Technologies discussed:

C# 7.0, Microsoft .NET Core/Framework, Tuples

Let's start the discussion with a focus on encapsulation and the benefits of properties versus fields. Consider, for example, an Arc value type representing a portion of the circumference of a circle. It's defined by the radius of the circle, the start angle (in degrees) of the first point in the arc, and the sweep angle (in degrees) of the last point in the arc, as shown in **Figure 1**.

Do Not Declare Fields That Are Public or Protected

In this declaration, Arc is a value type (defined using the keyword struct) with three public fields that define the characteristics of the Arc. Yes, I could've used properties, but I chose to use public fields in this example specifically because it violates the first guideline—***Do Not*** declare fields that are public or protected.

By leveraging public fields rather than properties, the Arc definition lacks the most basic of object-oriented design principles—encapsulation. For example, what if I decided to change the internal data structure to use the radius, start angle and arc length, for example, rather than the sweep angle? Doing so would obviously break the interface for Arc and all clients would be forced to make a code change.

Similarly, with the definitions of Radius, StartAngle and SweepAngle, I have no validation. Radius, for example, could be assigned a negative value. And while negative values for StartAngle and SweepAngle might be allowable, a value greater than 360 degrees wouldn't. Unfortunately, because Arc is defined using public fields, there's no way to add validation to protect against these values. Yes, I could add validation in version 2 by changing the fields to properties, but doing so would break the version compatibility of the

Arc structure. Any existing compiled code that invoked the fields would break at runtime, as would any code (even if recompiled) that passes the field as a by ref parameter.

Given the guideline that fields should not be public or protected, it's worth noting that properties, especially with default values, became easier to define than explicit fields encapsulated by properties, thanks to support in C# 6.0 for property initializers. For example, this code:

```
public double SweepAngle { get; set; } = 180;
```

is simpler than this:

```
private double _SweepAngle = 180;
public double SweepAngle {
  get { return _SweepAngle; }
  set { _SweepAngle = value; }
}
```

The property initializer support is important because, without it, an automatically implemented property that needs initialization would need an accompanying constructor. As a result, the guideline: "***Consider*** automatically implemented properties over fields" (even private fields) makes sense, both because the code is more concise and because you can no longer modify fields from outside their containing property. All this favors yet another guideline, "***Avoid*** accessing fields from outside their containing properties," which emphasizes the earlier-described data encapsulation principle even from other class members.

Figure 1 **Defining an Arc**

```
public struct Arc
{
  public Arc (double radius, double startAngle, double sweepAngle)
  {
    Radius = radius;
    StartAngle = startAngle;
    SweepAngle = sweepAngle;
  }
  public double Radius;
  public double StartAngle;
  public double SweepAngle;

  public double Length
  {
    get
    {
      return Math.Abs(StartAngle - SweepAngle)
        / 360 * 2 * Math.PI * Radius;
    }
  }

  public void Rotate(double degrees)
  {
    StartAngle += degrees;
    SweepAngle += degrees;
  }

  // Override object.Equals
  public override bool Equals(object obj)
  {
    return (obj is Arc)
      && Equals((Arc)obj);
  }
        // Implemented IEquitable<T>
  public bool Equals(Arc arc)
  {
    return (Radius, StartAngle, SweepAngle).Equals(
      (arc.Radius, arc.StartAngle, arc.SweepAngle));
  }

  // Override object.GetHashCode
  public override int GetHashCode() =>
    return (Radius, StartAngle, SweepAngle).GetHashCode();

  public static bool operator ==(Arc lhs, Arc rhs) =>
    lhs.Equals(rhs);
  public static bool operator !=(Arc lhs, Arc rhs) =>
    !lhs.Equals(rhs);
}
```

At this point lets return to the C# 7.0 tuple type ValueTuple<…>. Despite the guideline about exposed fields, ValueTuple<T1, T2>, for example, is defined as follows:

```
public struct ValueTuple<T1, T2>
  : IComparable<ValueTuple<T1, T2>>, ...
{
  public T1 Item1;
  public T2 Item2;

  // ...
}
```

What makes ValueTuple<…> special? Unlike most data structures, the C# 7.0 tuple, henceforth referred to as tuple, was not about the whole object (such as a Person or CardDeck object). Rather, it was about the individual parts grouped arbitrarily for transportation purposes, so they could be returned from a method without the bother of using out or ref parameters. Mads Torgersen uses the analogy of a bunch of people who happen to be on the same bus—where the bus is like a tuple and the people are like the items in the tuple. The Items are grouped together in a return tuple parameter because they are all destined to return to the caller, not because they necessarily have any other association to each other. In fact, it's likely that the caller will then retrieve the values from the tuple and work with them individually rather than as a unit.

By leveraging public fields rather than properties, the Arc definition lacks the most basic of object-oriented design principles—encapsulation.

The importance of the individual items rather than the whole makes the concept of encapsulation less compelling. Given that items in a tuple can be wholly unrelated to each other, there's often no need to encapsulate them in such a manner that changing Item1, for example, might affect Item2. (By contrast, changing the Arc length would require a change in one or both of the angles so encapsulation is a must.) Furthermore, there are no invalid values for the items stored within a tuple. Any validation would be enforced in the data type of the item itself, not in the assignment of one of the Item properties on the tuple.

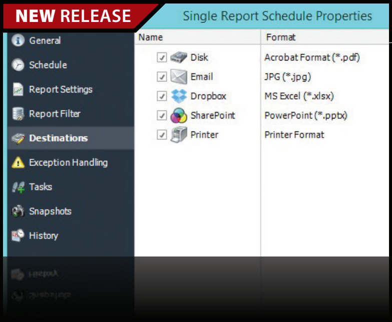
For this reason, properties on the tuple don't provide any value, and there's no conceivable future value they could provide. In short, if you're going to define a type whose data is mutable with no need for validation, you may as well use fields. Another reason you might want to leverage properties is to have varying accessibility between the getter and the setter. However, assuming mutability is acceptable, you aren't going to take advantage of properties with

ComponentSource®

/update/2018/06

www.componentsource.com

NEW RELEASE



Name	Format
✓ Disk	Acrobat Format (*.pdf)
✓ Email	JPG (*.jpg)
✓ Dropbox	MS Excel (*.xlsx)
✓ SharePoint	PowerPoint (*.pptx)
✓ Printer	Printer Format

PBRS (Power BI Reports Scheduler) | from $8,132.21

ChristianSteven

Data Driven Distribution for Power BI Reports & Dashboards.

- A comprehensive set of job (schedule) types gives you the power to automate delivery in Power BI
- Automate report delivery & send reports to printer, fax, folder, FTP, DropBox, SharePoint & email
- Contains powerful system event triggered, data-driven and business process workflow functions
- Respond instantly by firing off reports and automation scripts when an event occurs

BEST SELLER

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

LEADTOOLS® THE WORLD LEADER IN IMAGING SDKs

Powerful DICOM, PACS, and HL7 functionality.

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

BEST SELLER

DevExpress DXperience 17.2 | from $1,439.99

DevExpress

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

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

BEST SELLER

Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

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

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

mastercard VISA DISCOVER

GSA Schedule Contract GS-35F-0188R

differing getter/setter accessibility, either. This all raises another question—should the tuple type be mutable?

Do Not Define Mutable Value Types

The next guideline to consider is that of the mutable value type. Once again, the Arc example (shown in the code in **Figure 2**) violates the guideline. It's obvious if you think about it—a value type passes a copy, so changing the copy will not be observable from the caller. However, while the code in **Figure 2** demonstrates the concept of only modifying the copy, the readability of the code does not. From a readability perspective, it would seem the arc changes.

What's confusing is that in order for a developer to expect value copy behavior, they would have to know that Arc was a value type. However, there's nothing obvious from the source code that indicates the value type behavior (though to be fair, the Visual Studio IDE will show a value type as a struct if you hover over the data type). You could perhaps argue that C# programmers should know value type versus reference type semantics, such that the behavior in **Figure 2** is expected. However, consider the scenario in **Figure 3** when the copy behavior is not so obvious.

Notice that, in spite of invocation Arc's Rotate function, the Arc, in fact, never rotates. Why? This confusing behavior is due to the combination of two factors. First, Arc is a value type that causes it to be passed by value rather than by reference. As a result, invoking pie.Arc returns a copy of Arc, rather than returning the same instance of Arc that was instantiated in the constructor. This wouldn't be a problem, if it wasn't for the second factor. The invocation of Rotate is intended to modify the instance of Arc stored within pie, but in actuality, it modifies the copy returned from the Arc property. And that's why we have the guideline, "***Do Not*** define mutable value types."

Figure 2 **Value Types Are Copied So The Caller Doesn't Observe the Change**

```
[TestMethod]
public void PassByValue_Modify_ChangeIsLost()
{
  void Modify(Arc paramameter) { paramameter.Radius++; }
  Arc arc = new Arc(42, 0, 90);
  Modify(arc);
  Assert.AreEqual<double>(42, arc.Radius);
}
```

Figure 3 **Mutable Value Types Behave Unexpectedly**

```
public class PieShape
{
  public Point Center { get; }
  public Arc Arc { get; }

  public PieShape(Arc arc, Point center = default)
  {
    Arc = arc;
    Center = center;
  }
}

public class PieShapeTests
{
  [TestMethod]
  public void Rotate_GivenArcOnPie_Fails()
  {
    PieShape pie = new PieShape(new Arc(42, 0, 90));
    Assert.AreEqual<double>(90, pie.Arc.SweepAngle);
    pie.Arc.Rotate(42);
    Assert.AreEqual<double>(90, pie.Arc.SweepAngle);
  }
}
```

As before, tuples in C# 7.0 ignore this guideline and exposes public fields that, by definition, make ValueTuple<…> mutable. Despite this violation, ValueTuple<…> doesn't suffer the same drawbacks as Arc. The reason is that the only way to modify the tuple is via the Item field. However, the C# compiler doesn't allow the modification of a field (or property) returned from a containing type (whether the containing type is a reference type, value type or even an array or other type of collection). For example, the following code will not compile:

```
pie.Arc.Radius = 0;
```

Nor will this code:

```
pie.Arc.Radius++;
```

The tuple was introduced as a language feature to enable multiple return values without the complex syntaxrequired by out or ref parameters.

These statements fail with the message, "Error CS1612: Cannot modify the return value of 'PieShape.Arc' because it is not a variable." In other words, the guideline is not necessarily accurate. Rather than avoiding all mutable value types, the key is to avoid mutating functions (read/write properties are allowable). That wisdom, of course, assumes that the value semantics shown in **Figure 2** are obvious enough such that the intrinsic value type behavior is expected.

Do Not Create Value Types Larger Than 16 Bytes

This guideline is needed because of how often the value type is copied. In fact, with the exception of a ref or out parameter, value types are copied virtually every time they're accessed. This is true whether assigning one value type instance to another (such as Arc = arc in **Figure 3**) or a method invocation (such as Modify(arc) shown in **Figure 2**). For performance reasons, the guideline is to keep value type size small.

The reality is that the size of a ValueTuple<…> can often be larger than 128 bits (16 bytes) because a ValueTuple<…> may contain seven individual items (and even more if you specify another tuple for the eighth type parameter). Why, then, is the C# 7.0 tuple defined as a value type?

As mentioned earlier, the tuple was introduced as a language feature to enable multiple return values without the complex syntax required by out or ref parameters. The general pattern, then, was to construct and return a tuple and then deconstruct it back at the caller. In fact, passing a tuple down the stack via a return parameter is similar to passing a group of arguments up the stack for a method call. In other words, return tuples are symmetric with individual parameter lists as far as memory copies are concerned.

If you declared the tuple as a reference type, then it would be necessary to construct the type on the heap and initialize it with the Item values—potentially copying either a value or reference to the heap. Either way, a memory copy operation is required, similar to that of a value type's memory copy. Furthermore, at some later point in time when the reference tuple is no longer accessible, the garbage collector will need to recover the memory. In other words, a reference tuple still involves memory copying, as well as additional pressure on the garbage collector, making a value type tuple the more efficient option. (In the rare cases that a value tuple isn't more efficient, you could still resort to the reference type version, Tuple<…>.)

While completely orthogonal to the main topic of the article, notice the implementation of Equals and GetHashCode in **Figure 1**. You can see how tuples provide a shortcut for implementing Equals and GetHashCode. For more information, see "Using Tuples to Override Equality and GetHashCode."

Wrapping Up

At first glance it can seem surprising for tuples to be defined as immutable value types. After all, the number of immutable value types found in .NET Core and the .NET Framework is minimal, and there are long-standing programming guidelines that call for value types to be immutable and encapsulated with properties. There's also the influence of the immutable-by-default approach characteristic to F#, which pressured C# language designers to provide a shorthand to either declare immutable variables or define immutable types. (While no such shorthand is currently under consideration for C# 8.0, read-only structs were added to C#7.2 as a means to verify that a struct was immutable.)

In the end, guidelines are just that, guidelines.

However, when you delve into the details, you see a number of important factors. These include:

- Reference types impose an additional performance impact with garbage collection.
- Tuples are generally ephemeral.
- Tuple items have no foreseeable need for encapsulation with properties.
- Even tuples that are large (by value type guidelines) don't have significant memory copy operations beyond that of a reference tuple implementation.

In summary, there are plenty of factors that favor a value type tuple with public fields in spite of the standard guidelines. In the end, guidelines are just that, guidelines. Don't ignore them, but given sufficient—and I would suggest, explicitly documented—cause, it's OK to color outside the lines on occasion.

Using Tuples to Override Equality and GetHashCode

In the past, the implementation of Equals and GetHashCode were fairly complex, yet the actual code is generally boilerplate. For Equals, it's necessary to compare all the contained identifying data structures, while avoiding infinite recursion or null reference exceptions. For GetHashCode, it's necessary to combine the unique hash code of each of the non-null contained identifying data structures in an exclusive OR operation. With C# 7.0 tuples, this turns out to be quite simple, as is demonstrated in **Figure 1**.

For Equals, one member can check that the type is the same, while a second member groups each of the identifying members into a tuple and compares them to the target parameter of the same type, like this:

```
public override bool Equals(object obj) =>
  return (obj is Arc)
    && Equals((Arc)obj);

public bool Equals(Arc arc) =>
  return (Radius, StartAngle, SweepAngle).Equals(
    (arc.Radius, arc.StartAngle, arc.SweepAngle));
```

You might argue that the second function could be more readable if each identifying member were explicitly compared instead, but I leave that for the reader to arbitrate. That said, note that internally the tuple (System.ValueTuple<…>) uses EqualityComparer<T>, which relies on the type parameters implementation of IEquatable<T>, which only contains a single Equals<T>(T other) member. Therefore, to correctly override Equals, you need to follow the guideline: "***Do*** implement IEquatable<T> when overriding Equals." That way your own custom data types will leverage your custom implementation of Equals rather than Object.Equals.

Perhaps the more compelling of the two overloads is GetHashCode and its use of the tuple. Rather than engage in the complex gymnastics of an exclusive OR operation of the non-null identifying members, you can simply instantiate a tuple of all identifying members and return the GetHashCode value for the said tuple, like so:

```
public override int GetHashCode() =>
  return (Radius, StartAngle, SweepAngle).GetHashCode();
```

Nice!!!

For more information on the guidelines for both defining value types and overriding Equals and GetHashCode, check out chapters 9 and 10 in my Essential C# book: "Essential C# 7.0" (IntelliTect.com/EssentialCSharp), which is expected to be out in May. ■

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 has been 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# 6.0 (5th 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 following Microsoft technical experts for reviewing this article: *Kevin Bost, Eric Lippert, Mads Torgersen*

Developing Perspective

Visual Studio Live! (VSLive!™) returns to Boston, June 10 – 14, 2018, with 5 days of practical, unbiased, Developer training, including NEW intense hands-on labs. Join us as we dig into the latest features of **Visual Studio 2017, ASP.NET Core, Angular, Xamarin, UWP** and more. Code with industry experts AND Microsoft insiders while developing perspective on the Microsoft platform.

VSLive! 2001

VSLive! 2017

BACK BY POPULAR DEMAND: Pre-Con, Hands-On Labs!

Sunday, June 10
Choose from:
› ASP.NET Core 2 and EF Core 2
› Xamarin and Xamarin.Forms

REGISTER NOW

vslive.com/boston

Register to Code With Us Today!

Use Promo Code MSDN

SILVER SPONSOR JetBrains

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

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

AGENDA AT-A-GLANCE

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

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

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

Speakers and sessions subject to change

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/boston

Introducing Azure Blockchain Workbench

Stefano Tempesta

Azure Blockchain Workbench (aka.ms/abcworkbench) is the latest step in Microsoft's journey to enable customers to adopt blockchain technologies and get started with Azure Blockchain. To provide context, Azure Blockchain is a collection of Azure services and capabilities designed to help enterprises create and deploy a new class of applications for sharing business processes and data with multiple, semi-trusted organizations. Currently customers can deploy these services into their Azure subscriptions and integrate them with blockchains available on the Azure Marketplace. Now, with Azure Blockchain Workbench, the heavy lifting is done for them, so they can focus less on scaffolding and more on logic and smart contracts.

Azure Blockchain Workbench orchestrates several Azure services around popular blockchain digital ledgers and into a reference architecture that can be used to build blockchain-based applications. It's a free, easy-to-use tool with a simplified interface that enables users to create end-to-end decentralized applications leveraging all of Azure Blockchain's capabilities. It comes equipped with sample cross-organizational workflows and smart contracts, as well as an out-of-the-box UI that customers can use to drastically reduce development costs and accelerate proofs of concept.

This article discusses:

- Deploying and configuring Azure Blockchain Workbench
- Creating and deploying business logic in the form of smart contracts to drive scenario flows
- Using Azure Service Bus to extend Azure Blockchain Workbench

Technologies discussed:

Azure Blockchain, Azure Service Bus

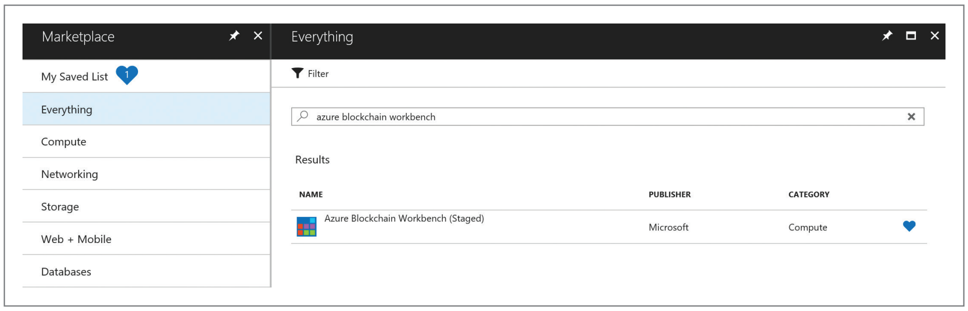
If you're new to blockchain, I recommend you read Jonathan Waldman's outstanding article, "Blockchain Fundamentals" (msdn.com/magazine/mt845650), in the March 2018 issue of *MSDN Magazine*. Also, make a point to watch the Microsoft Mechanics video, "Intro to Blockchain and Azure Blockchain Workbench," at aka.ms/workbenchintro.

Introducing Azure Blockchain Workbench

In much the same way that applications today interact with databases, decentralized applications running on a blockchain communicate and execute logic against a specific digital ledger, such as Ethereum or Hyperledger. A digital ledger network consists of a peer-to-peer decentralized network of nodes. These nodes maintain a copy of the data store and run a virtual machine to support arbitrary computation against the ledger, while maintaining consensus. Smart contracts are the mechanism that allows for this complicated computation on the network, similar to stored procedures on a traditional relational database.

Despite the potential to leverage blockchain in a growing variety of business processes, there are obstacles to overcome before enterprises can adopt blockchain technologies. Lack of governance, network performance and scalability are often cited as challenges that impact the adoption of blockchain in the enterprise space.

Building a complete solution running on blockchain typically requires provisioning infrastructure and developing a client application, writing and deploying a gateway API, implementing support for off-chain storage, writing logs and reports, and integrating identity and key vault services into the solution. There's a common set of challenges related to blockchain app development that are addressed by Azure Blockchain Workbench, which dramatically reduces the amount of time needed to build a blockchain solution.



azure blockchain workbench

Results

NAME	PUBLISHER	CATEGORY
Azure Blockchain Workbench (Staged)	Microsoft	Compute

Figure 1 **The Azure Blockchain Workbench in the Azure Marketplace**

Azure Blockchain Workbench provides code assets and ARM template-driven deployment for all the scaffolding needed for blockchain POCs, including the blockchain network, a gateway API, a responsive Web application, Azure Active Directory integration, Azure Key Vault integration, a SQL DB configured for collecting on-chain data, and a set of supporting services for data hashing and signing. The tool also makes it possible to create a Web front end without writing any code. It uses metadata provided for smart contracts to dynamically deliver a contextual UX for participants. Because the framework populates SQL DB as an off-chain store, it enables an organization to leverage existing skills and tools to light up additional capabilities.

The result: Azure Blockchain Workbench reduces time and cost for proof-of-concept (POC) projects. It enables customers to focus on creating innovative applications that demonstrate the potential of blockchain, by spending less time and resources on integration tasks that are required to stand up a basic POC.

Azure Blockchain Workbench is available on the Azure Marketplace (aka.ms/tryworkbench), and a video of Workbench in action is available by the Microsoft Mechanics team at aka.ms/workbenchdemo.

Deploy Azure Blockchain Workbench

The deployment of Azure Blockchain Workbench requires the following prerequisite steps:

1. An Azure Active Directory (Azure AD) tenant to host the necessary Azure AD and Key Vault apps.
2. Registration of an Azure AD app and a Key Vault value to be referenced during the deployment of Azure Blockchain Workbench.

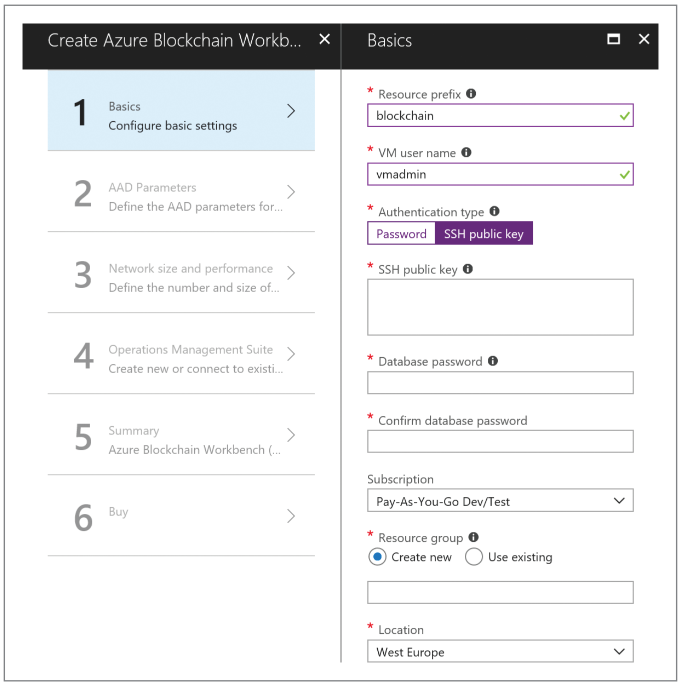
You may want to create a new Azure AD tenant for setting up test users or registering applications in a different Azure AD tenant than your corporate Azure AD tenant. Detailed instructions on these configuration steps are reported in the official Azure Blockchain Workbench documentation at aka.ms/workbenchdocs.

Once the prerequisite steps have been completed, the Workbench can be deployed. Sign in to the Azure portal and add a new resource from the Azure Marketplace, shown in **Figure 1**. Search for and select Azure Blockchain Workbench to start the provisioning process.

This solution template is designed to ease deployment and integration of the services needed to build an application on a blockchain network. With a handful of user inputs and a single-click deployment through the Azure portal, you can deploy a blockchain ledger along with the relevant Azure services needed to build an end-to-end blockchain application, packaged with a sample Web UI. The generated application includes a Web application, REST API, off-chain storage and the like. Rather than spending hours building out and configuring the services needed to integrate with a blockchain network, the Azure team has automated these time-consuming pieces so you can focus on building out your scenarios. Through the Azure Blockchain Workbench Web application, you can generate an end-to-end UX for your business workflows based on smart contracts.

Provisioning Azure Blockchain Workbench requires six steps, with the first two establishing configuration parameters, and the next two addressing network capacity and monitoring. Step 1 requires some basic settings, depicted in **Figure 2**, which include:

- A unique identifier as a prefix for naming all Azure resources provisioned as part of this template.
- The admin username for all provisioned virtual machines (VMs).
- An SSH public key used for connecting to the VMs. Copy and paste an RSA public key in the single-line format (starting with "ssh-rsa") or the multi-line PEM format. You can generate SSH keys using ssh-keygen on Linux and macOS, or PuTTYGen on Windows.
- A password to protect access to the database included as part of the Workbench deployment.
- The Azure subscription you wish to use for your deployment. Remember, use of Azure Blockchain Workbench is free of any license cost, but you pay for the provisioned Azure computing, storage and network resources.


Create Azure Blockchain Workb...
Basics
1 Basics
Configure basic settings
2 AAD Parameters
Define the AAD parameters for...
3 Network size and performance
Define the number and size of...
4 Operations Management Suite
Create new or connect to existi...
5 Summary
Azure Blockchain Workbench (...
6 Buy

* Resource prefix	blockchain
* VM user name	vmadmin
* Authentication type	Password [✓ SSH public key]
* SSH public key	
* Database password	
* Confirm database password	
Subscription	Pay-As-You-Go Dev/Test
* Resource group	[✓ Create new] Use existing
* Location	West Europe

Figure 2 **Getting Started Provisioning Azure Blockchain Workbench**

- The Resource group to use to group all these resources together. I recommend creating a new resource group for your Blockchain Workbench solution.
- The region to which you wish to deploy the resources.

Step 2 requires you to have completed the prerequisite steps of registration of the blockchain client app in Azure AD, as you'll need the Registration Application ID and key to enter when requested. You also need to specify the Tenant domain name obtained as part of the registration of the client app in Azure AD.

In Step 3 you can specify the number of nodes in the provisioned blockchain ledger and the size of the allocated VMs. Step 4 is for optionally opting in to the Operations Management Suite (OMS) for monitoring the deployed solution.

Steps 5 and 6 are simply a summary of the resources being provisioned and your acknowledgment that you understand that there is a cost implication. Once the deployment of the Workbench has completed, you'll see a new Resource Group with the specified name. Some resources deployed here, such as Application Insights, will allow you to get more information about the state of the Workbench, including details about the deployed VMs and networks.

As I noted earlier, Azure Blockchain Workbench leverages Azure AD for authentication, access control and workflow persona identification. Only users specified in the referenced Azure AD will be able to authenticate and use the deployed resources. In addition, users must be associated with a specific user group in order to interact and perform actions. It's therefore necessary to manage user accounts interacting with Azure Blockchain Workbench and assign them the required permissions.

Deploying a Smart Contract

Once deployed, you can access Azure Blockchain Workbench by navigating to its URL. You'll see an Azure AD-backed login experience where you can enter your work or personal Microsoft account credentials to access the application.

Smart contracts are written in blockchain stack-specific languages. For example, Solidity is used for Ethereum, while Go is used for Hyperledger Fabric.

You can now start leveraging smart contracts, which contain business logic that drives different scenario flows. This approach provides the immutability, deterministic execution and transparency required in untrusted environments. Smart contracts are written in blockchain stack-specific languages. For example, Solidity is used for Ethereum, while Go is used for Hyperledger Fabric. A smart contract is deployed to all nodes on the blockchain. During execution, the information it conveys is also replicated to all the nodes on the network.

Figure 3 **LexingtonBase Contract**

```
contract LexingtonBase {
  event LexingtonContractCreated(string contractType, address originatingAddress);
  event LexingtonContractUpdated(string contractType, string action,
    address originatingAddress);

  string internal ContractType;

  function LexingtonBase(string contractType) internal {
    ContractType = contractType;
  }

  function ContractCreated() internal {
    LexingtonContractCreated(ContractType, msg.sender);
  }

  function ContractUpdated(string action) internal {
    LexingtonContractUpdated(ContractType, action, msg.sender);
  }
}
```

Figure 4 **Asset Transfer Smart Contract**

```
contract AssetTransfer is LexingtonBase('AssetTransfer')
{
  enum AssetState { Created, Active, OfferPlaced, PendingInspection,
    Inspected, Appraised,
    NotionalAcceptance, BuyerAccepted, SellerAccepted, Accepted,
      Complete, Terminated }
  address public Owner;
  string public Description;
  uint public AskingPrice;
  AssetState public State;

  address public Buyer;
  uint public OfferPrice;
  address public Inspector;
  address public Appraiser;

  function AssetTransfer(string description, uint256 price)
  {
    Owner = msg.sender;
    AskingPrice = price;
    Description = description;
    State = AssetState.Active;
    ContractCreated();
  }
}
```

The main goal of Azure Blockchain Workbench is to stand up the scaffolding around the blockchain application, so users can focus on creating the smart contract-based business logic. Once a smart contract is instantiated in the Workbench, it's written to the blockchain, and subsequent updates to the smart contract are recorded on the blockchain. After the Azure Blockchain Workbench administrator has deployed smart contracts and completed user assignments for the smart contracts, other users can interact with the blockchain application and engage in the smart contract workflows.

Let's start by deploying any of the existing demo smart contracts. After signing in successfully, you should see an Admin link on the top right. Click on the Admin link to access the Administrator Dashboard. There are currently three actions available to the Administrator in the Workbench. From here you can assign users, deploy demo contracts, and deploy custom contracts.

Clicking on Deploy Demo Contract in the Administrator Dashboard brings you to a list of all the demo contracts that can be provided. The current release of Blockchain Workbench contains a set of demo smart contracts and configuration files to get you started.

Imaging SDK for Winforms, WPF, and Web Development

GdPicture.NET

- Scanning
- OCR
- 100+ Formats
- Image Cleanup
- Annotations
- Barcodes
- Document Compression
- MICR
- Form Processing
- Thumbnails
- Viewer Control
- PDF
- Bookmarks
- Image Processing
- Color Detection
- Printing
- DICOM
- TIFF
- DOCX
- Metadata Support
- Document Conversion

Leverage your apps. with GdPicture.NET Imaging Toolkit

60-day Free Trial
Support Included

www.gdpicture.com

GdPicture.NET is an ORPALIS imaging technologies product

The Workbench documentation provides more information about the specific scenarios and a step-by-step walk-through of the UI. There are also detailed instructions on how the demo smart contracts and associated configuration files were created, and how to set up the right user accounts in Azure AD to enable this scenario.

Uploading a Custom Contract

You can also create custom contracts in any of the programming languages supported by the blockchain platform of reference, and deploy them in Azure Blockchain Workbench. All Azure Blockchain Workbench requires is three files that define business logic, interactions and visualizations of the implemented workflows. These are:

Smart Contract: To begin, you must create a smart contract that contains the business logic for the scenario. In the following example, the smart contract is targeting the Ethereum blockchain, so the contracts are written in the Solidity programming language.

Configuration: In the configuration file, users map properties, actions and blockchain protocol information for the smart contract. You also define what parameters are required from the participants for the instantiation and actions on the smart contract.

UI Configuration: In the configuration UI files, you define the UI for role-based access control at each state of the smart contract, such as restricting which personas can take specific actions. In this file, the user maps states and personas, and specifies state definitions, percentage complete at each state of the contract, and eligible actions at each state.

Smart Contract Creation

The contract definition contains a few components that must be specified to work with Azure Blockchain Workbench. Before defining the contract details specific to the scenario such as states, participants and the functions that describe the logic behind each action, you have to implement a LexingtonBase contract. The actual contract with the business logic for the business scenario will inherit from the LexingtonBase contract (which is equivalent to a class in C#), as shown in **Figure 3**.

Each contract should have a state that represents the current state of the contract, addresses for participants involved in the smart contract, data that's stored in the contract, functions defining the business logic for different actions and a constructor for when the smart contract is instantiated. **Figure 4** provides an example of the constructor and properties of the Asset Transfer smart contract.

Be sure that your class name and constructor name match. You can find the sample source code in Solidity language for this smart contract, along with the JSON configuration files, on my GitHub repository at bit.ly/2HJPcET.

Configuration File Creation

The configuration file defines the main properties and parameters for the smart contract. This includes listing out all properties and types, as well as defining a constructor indicating what parameters the user needs to input to instantiate the smart contract. **Figure 5** shows an example of the Asset Transfer properties and constructor from the configuration file.

In addition, the user must enumerate and provide detail for all possible actions, including specifying input parameters for each action. **Figure 6** shows an example of the MakeOffer action specification from the Asset Transfer configuration file, which requires the user to input an inspector, appraiser and an offer price.

Last, include the blockchain configuration information, as follows:

```
"Chains": {
    "Ethereum": {
      "Type": "Ethereum",
      "Version": "1.0",
      "Location": "AssetTransfer.sol",
      "TypeName": "AssetTransfer",
      "ActionOverrides": {},
      "PropertyOverrides": {}
    }
  }
```

Figure 5 **Asset Transfer Properties and Constructor**

```
"Properties": {
    "State": {
      "Type": "state"
    },
    "Owner": {
      "Type": "user"
    },
    "Description": {
      "Type": "string"
    },
    "AskingPrice": {
      "Type": "money"
    },
    "Buyer": {
      "Type": "user"
    },
    "OfferPrice": {
      "Type": "money"
    },
    "Inspector": {
      "Type": "user"
    },
    "Appraiser": {
      "Type": "user"
    }
  },
"Constructor": {
    "description": {
      "Type": "string"
    },
    "price": {
      "Type": "money"
    }
  },
```

Figure 6 **MakeOffer Action Specification**

```
"MakeOffer": {
    "Parameters": {
      "inspector": {
        "Type": "user"
      },
      "appraiser": {
        "Type": "user"
      },
      "offerPrice": {
        "Type": "money"
      }
    }
  },
```

UI Configuration File Creation

Users define the UI details for each action in the UI configuration file. This includes details on the users, such as the initiator or participant role and persona mapping, state mappings, as well as a constructor indicating text to be displayed when a user wants to instantiate the smart contract. **Figure 7** shows an example of the Asset Transfer properties and constructor from the UI configuration file.

In addition, the user should enumerate all possible states and include details for each state specifying actions that can be taken at each step per persona, as well as a percentage-complete value to give users a visualization of progress through the smart contract. For example, the Asset Transfer contract UI configuration file shows that once the state shows an offer has been placed, only owners and buyers can take specified actions, as shown in the code in **Figure 8**.

Once you've created your custom smart contract and associated configuration files, the next task is to deploy the smart contract. After selecting Upload Custom Contract in the Administrator Dashboard of Azure Blockchain Workbench, you'll see a page to upload your custom contract and configuration files. There are three files that you need to provide. Select the smart contract file and the two config files created earlier, as indicated in the Upload Contract screen.

User Assignment

Once a smart contract (demo or custom) is deployed, users with initiator personas can create new contract instances. To create an instance of a particular contract, the user must have a persona associated with that contract. Depending on the specification of the smart contract, not every persona may have rights to create a contract. As an administrator, you can assign a user to a contract and specify their role (the persona) in the business process. Note that before an administrator can perform any user assignment actions, there must be at least one contract (demo or custom) deployed. Once a contract is uploaded, the next step is to complete the user assignment for the contract. Administrators can assign users to smart contracts from the User Assignment screen in the Azure Blockchain Workbench Administrator Dashboard.

Once assigned to contracts, users can participate in the smart contract workflows by signing in to Azure Blockchain Workbench to create contracts. Say that the user wants to generate a new asset transfer. He or she must create a new contract by clicking Create New Contract on the upper right of the view and enter the details for the contract (this view will vary based on your smart contract) and submit. The screen is automatically generated, based on the metadata provided as part of the Smart Contract definition.

After a contract instance is created, a user can drill down into the details to view available actions, given the current state of the contract.

Figure 7 **InitiatingRoles**

```
"InitiatingRoles": [
    "Admin",
    "User"
  ],
  "Personas": {
    "Owner": {
      "Role": "Initiator",
      "IsInitiator": true,
      "PropertyMapping": "Owner"
    },
    "Buyer": {
      "Role": "Participant",
      "PropertyMapping": "Buyer"
    },
    "Appraiser": {
      "Role": "Participant",
      "PropertyMapping": "Appraiser"
    },
    "Inspector": {
      "Role": "Participant",
      "PropertyMapping": "Inspector"
    }
  },
  "StateProperty": "State",
  "StateMapping": {
    "Created": 0,
    "Active": 1,
    "OfferPlaced": 2,
    "PendingInspection": 3,
    "Inspected": 4,
    "Appraised": 5,
    "NotionalAcceptance": 6,
    "BuyerAccepted": 7,
    "SellerAccepted": 8,
    "Accepted": 9,
    "Complete": 10,
    "Terminated": 11
  },
"Constructor": {
    "DisplayName": "Create Asset Transfer",
    "Description": "Description of asset transfer"
  },
```

Figure 8 **OfferPlaced Code**

```
"OfferPlaced": {
    "PercentComplete": 30,
    "Style": "Success",
    "Actions": {
      "Owner": {
        "AcceptOffer": {},
        "Reject": {},
        "Terminate": {}
      },
      "Buyer": {
        "RescindOffer": {}
      }
    }
  }
```

Extending Azure Blockchain Workbench

At the core of Azure Blockchain Workbench is Azure Service Bus, enabling an extensible and pluggable model that allows multiple distributed ledger technologies, storage and database offerings to be used as part of the blockchain solution. There are also opportunities to integrate other services with the Workbench to extend functionality, such as with Azure Logic Apps, Web APIs, Notification Hubs and the like. Off-chain storage of data will allow for post-processing storage and analytics scenarios, with options such as with Power BI, Azure Machine Learning, HD Insight, Azure Data Lake and other services where contract data is shared.

Blockchain has the potential to extend digital transformation beyond a company's walls and into the processes it shares with suppliers, customers and partners. As I've shown, at its core a blockchain is both a computing and data structure that can be used to create a digital transaction ledger that, instead of resting with a single provider, is shared among a distributed network of computers. The result is a more transparent and verifiable system that will change the way you think about exchanging value and assets, enforcing contracts, and sharing data.

Microsoft is committed to bringing blockchain to the enterprise and bringing the full benefits of Azure to bear for developers and organizations looking to build distributed applications. The goal is to help companies thrive in this new era of secure, multi-party computation by delivering scalable platforms and services that any company—from ledger startups and retailers to health providers and global banks—can use to improve shared business processes. Azure Blockchain Workbench is part of an ecosystem of services in Azure, along with the announced Azure Confidential Computing and the Confidential Consortium Blockchain Framework, that helps bridge the gap between the blockchain world and enterprise requirements for governance, security and scalability. Watch this space for additional development in the near future! ■

Stefano Tempesta *is a Microsoft Regional Director and MVP, as well as chapter leader for CRMUG in Switzerland, the largest community of Dynamics 365/CRM experts in the world. Tempesta is an instructor of courses about Dynamics 365, blockchain and machine learning, and a regular speaker at international IT conferences, including Microsoft Ignite and Tech Summit. He founded 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 experts for reviewing this article: *Marc Mercuri, JT Rose*

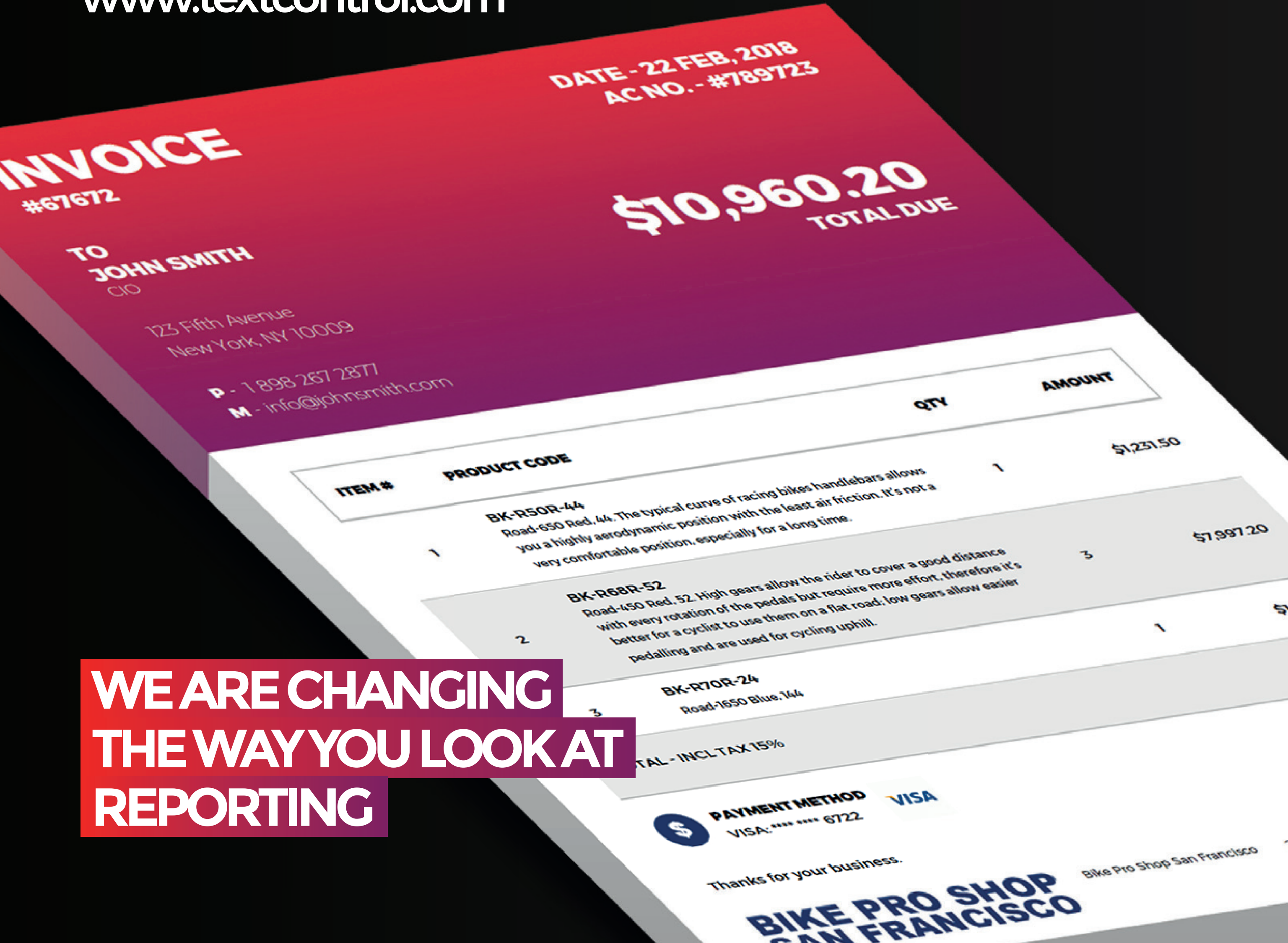
TEXT CONTROL

TX Text Control X15

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

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

Download a free trial at
www.textcontrol.com



INVOICE
#67672

DATE - 22 FEB, 2018
AC NO. - #789723

$10,960.20
TOTAL DUE

TO
JOHN SMITH
CIO

123 Fifth Avenue
New York, NY 10009

P - 1 898 267 2877
M - info@johnsmith.com

ITEM #	PRODUCT CODE	QTY	AMOUNT
1	BK-R50R-44 Road-650 Red, 44. The typical curve of racing bikes handlebars allows you a highly aerodynamic position with the least air friction. It's not a very comfortable position, especially for a long time.	1	$1,231.50
2	BK-R68R-52 Road-450 Red, 52. High gears allow the rider to cover a good distance with every rotation of the pedals but require more effort, therefore it's better for a cyclist to use them on a flat road; low gears allow easier pedalling and are used for cycling uphill.	3	$7,997.20
3	BK-R70R-24 Road-1650 Blue, 144	1	$1

TAL - INCL TAX 15%

PAYMENT METHOD
VISA: **** **** 6722
VISA

Thanks for your business.

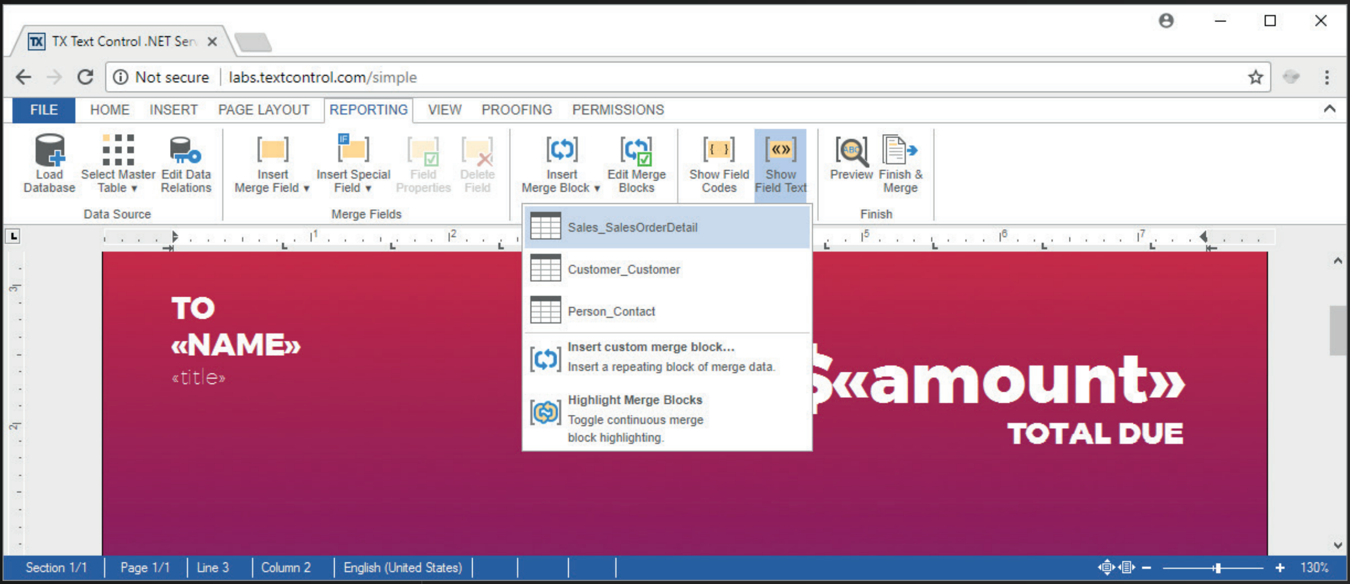
BIKE PRO SHOP SAN FRANCISCO
Bike Pro Shop San Francisco

WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio Partner

TX Text Control .NET Server for ASP.NET

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



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

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

TEXT CONTROL

Monitoring Databricks Jobs with Application Insights

Joseph Fultz and Ryan Murphy

We work on a team that focuses on data and analytics for large companies that want to implement or migrate their solutions to the cloud. These efforts come with the obvious work of optimizing and reengineering applications to various degrees to ensure they take advantage of what the cloud offers. As great as those efforts can be for the application itself, there are additional challenges for organizations just starting their cloud journey, as they must also do all of the work that goes along with extending their operational capabilities to the cloud. And, as new technologies emerge and evolve, these must be folded into the existing operational infrastructure, as well. This is one of the challenges that exists with Spark- and Apache Hadoop-based solutions. Yes, Apache Ambari is there to provide a nice dashboard and has an API to expose metrics, but many organizations already have an investment in and a good understanding of other monitoring and dashboarding solutions, such as Azure Application Insights.

This article discusses:

- Planning for Application Insight telemetry
- Adding Applications Insight to Azure Databricks Clusters
- Instrumenting Databricks job code
- Configuring analytics and alerts

Technologies discussed:

Application Insights, Azure Databricks

Imagine a WebJob that pulls messages from Azure Event Hubs, does some initial validation, and then drops them into Azure Storage, at which point that data is processed through several Spark jobs, as shown in **Figure 1**. The goal is to have a single runtime dashboard that can provide information that shows not only what's happening, but also process- and business-specific information while it's in flight. Additionally, it would be great to be able to track the flow of that information as a holistic process and see details on its constituent processes.

Sure, you can see default metrics for WebJobs in Application Insights and some information from the Spark jobs in Ambari—and roll them all up with Azure Log Analytics for post hoc insights. However, we don't want to see two separate processes with four steps each. We want to see the process as a whole and we want runtime insights and runtime alerts.

In this article, we'll walk through considerations and planning for bringing the full project together using Application Insights. Additionally, we'll be using the Azure Databricks flavor of Spark as it has a nice set of features that help us more easily develop and operationalize our workflow.

Planning for Application Insights Telemetry

We won't be covering core concepts, but for a primer on these concepts take a look through the online documentation at bit.ly/2FYOCyp. Also, Victor Mushkatin and Sergey Kanzhelev wrote a good article about optimizing telemetry data collection, "Optimize Telemetry

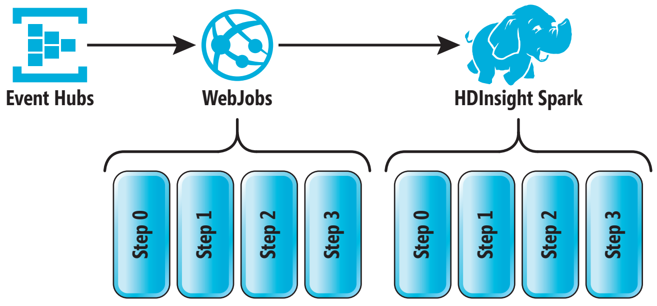


Figure 1 **Single Solution, Separate Processes, Separate Steps**

with Application Insights" (msdn.com/magazine/mt808502). Here, we'll focus on organizing our notebooks and jobs to facilitate proper tracking in the form of the operation, event and data we send from our Databricks jobs.

In Databricks, you can define a job as the execution of a notebook with certain parameters. **Figure 2** illustrates a couple of basic approaches to organizing work in a Databricks Notebook.

Figure 2 shows two simple possibilities in which one job is defined as a single notebook with a number of code blocks or functions that get called while the other job displays a control notebook that orchestrates the execution of child notebooks, either in sequence or in parallel. This is not, by any means, the only organization that can be used, but it's enough to help illustrate how to think about correlation. How you go about organizing the notebooks and code is certainly a worthwhile topic and is highly variable depending on the size and nature of the job. For a little more depth on Databricks Notebook Workflow, take a look at the blog post, "Notebook Workflows: The Easiest Way to Implement Apache Spark Pipelines" (bit.ly/2HOqvTj).

Notice that notebook organization has been aligned with discrete operations that can be used to group event reporting in Application Insights. In Application Insights, correlation is accomplished via two properties: Operation Id and Parent Operation Id. As seen in **Figure 2**, we wish to capture all of the discrete events and metrics within a code block or separate notebook under the context of a single operation, which is done by using a distinct operation Id for each section. Additionally, we'd like to see those separate large operation blocks as part of a whole, which we can do by setting the context's parent operation Id to the same value for all metrics reporting in each operation. The parent operation Id can also be passed in from an outside trigger for the job, which would then provide a mechanism to link all of the discrete operations from the previous process and the Azure Databricks job as part of a single gestalt operation identified by the parent operation Id in Application Insights.

We've depicted a couple scenarios here. The key point is that you should consider how you want to organize your operations, events and metrics as part of the overall job organization.

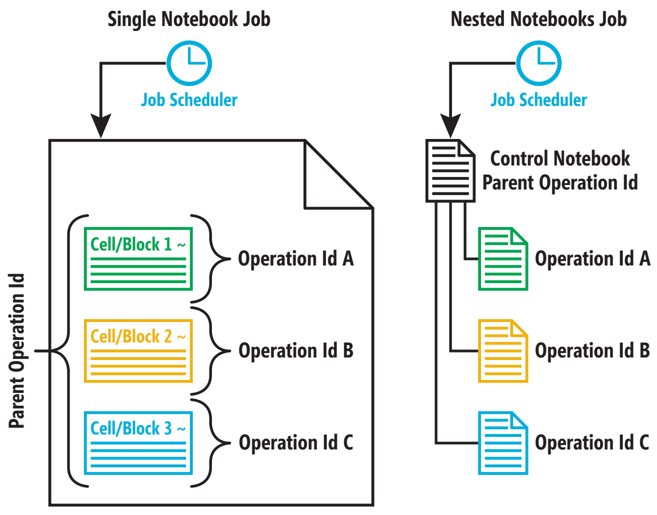


Figure 2 **Basic Organization Options for a Databricks Notebook Job**

Notice that notebook organization has been aligned with discrete operations that can be used to group event reporting in Application Insights.

Adding Application Insights to the Environment

In order to get the environment ready, you need to install the Python Application Insights library on the cluster, grab some configuration settings and add a bit of helper code. You can find Application Insights on pypi (pypi.python.org/pypi/applicationinsights/0.1.0). To add it to Databricks, simply choose a location in your workspace (we created one named Lib) and right-click and choose Create, then Library. Once there, you can enter the pypi application name and Databricks will download and install the package. The last thing you'll have to decide is whether or not you want to attach the library to all clusters automatically.

In the attempt to reduce the amount of code to add to each notebook, we've added an include file that has a couple of helper functions:

```
def NewTelemetryClient (applicationId, operationId="",
  parentOperationId=""):
  tc = TelemetryClient(instrumentationKey)
  tc.context.application.id = applicationId
  tc.context.application.ver = '0.0.1'
  tc.context.device.id = 'Databricks notebook'
  tc.context.operation.id = operationId
  tc.context.operation.parentId = parentOperationId
  return tc
```

This code contains a factory function named NewTelemetryClient to create a telemetry client object, set some of the properties and return the object to the caller. As you can see, it takes a parent operation Id and an operation Id. This initializes the object, but note that if you need to change the operation Id, you'll have to do it in the job notebook directly. Also worth noting is that the TelemetryClient constructor takes an instrumentation key, which can be found in the properties blade of the Application Insights instance you wish to use. We've statically assigned a few values that are needed for the example, but the TelemetryClient context object has many child objects and properties that are available. If

Figure 3 **Notebook Initialization Code**

```
baseRatingsFile = dbutils.widgets.get("baseRatingsFile")
newRatingsFile = dbutils.widgets.get("newRatingsFile")
trainOperationId = dbutils.widgets.get("trainOperationId")
parentOperationId = dbutils.widgets.get("parentOperationId")
maxIterations =  int(dbutils.widgets.get("maxIterations"))
numFolds = int(dbutils.widgets.get("numFolds"))
numUserRecommendations = int(
  dbutils.widgets.get("numUserRecommendations"))
predictionFilePath = dbutils.widgets.get("predictionFilePath")

if trainOperationId == "":
  trainOperationId = NewCorrelationId()

if parentOperationId == "":
  parentOperationId = NewCorrelationId()

#setup other needed variables
telemetryClient = NewTelemetryClient("PredictionExample",
  trainOperationId, parentOperationId)

telemetryClient.context.operation.name = "Train Model"
```

you needed to initialize other values, this would be the place to do it. Separating out the factory function keeps the clutter down and also eases implementation for the developer converting the notebook from a sandbox prototype kind of code to an enterprise job kind of implementation.

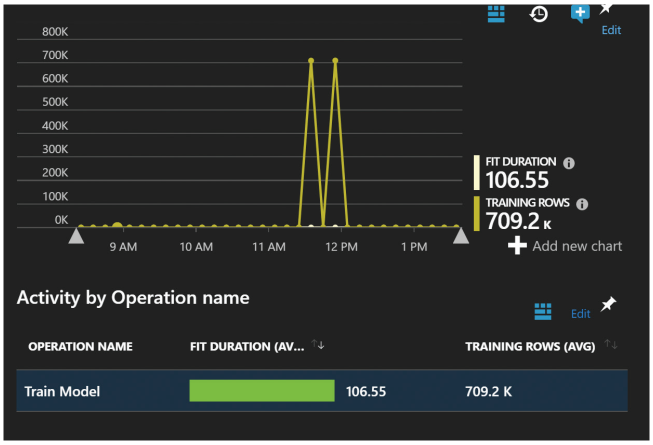
With the library added to the cluster and the setup notebook defined, we simply need to add a line at the top of the job notebook to run the setup and then create a starter telemetry object. We'll issue a %run command at the top of the notebook:

```
%run ./AppInsightsSetup
```

The fully prepared data set is persisted to long-term Blob Storage and an aggregated subset is sent to our RDBMS, Azure SQL Database.

In the subsequent cell we'll simply instantiate a new instance of the TelemetryClient object.

Figure 3 shows the code from the prediction example we created. There are several things to take note of here. First, we're passing in a number of variables to the notebook that are sent as part of the job initialization, which is done via the dbutils.widgets object provided as part of the Databricks environment. Because we need a couple of IDs for the parent operation and the discrete operation, we'll go ahead and check those and, if they're empty, create and assign new UUIDs. Assigning the arbitrary IDs in this case is mostly to make it easier to run interactively. However, other approaches could be taken, such as encapsulating the job notebook's code into a series of functions and running tests by calling the parent function with a specific ID. Both work sufficiently well for our purposes here. The last thing we assign is an operation name, which eventually shows up in Application Insights as something you can use to view and group by, as seen in **Figure 4**.



Activity by Operation name

OPERATION NAME	FIT DURATION (AV...	TRAINING ROWS (AVG)
Train Model	106.55	709.2 K

Figure 4 **Operation Name in Application Insights**

Looking at **Figure 3**, you can see that the operation name was assigned the value of Train Model. **Figure 4** depicts it in a grid of data after it was chosen as the grouping mechanism for the data. As we run more jobs through and assign differing operation names, we'll be able to see those show up in the view, as well. With those things in place, we're in good shape to work on instrumenting our job code to capture events and metrics.

Instrumenting Databricks Job Code

Let's walk through an example that uses Application Insights to monitor a typical data-engineering job in Databricks. In this scenario, we're using publicly available data from Fannie Mae (bit.ly/2AhL5sS) and will take raw source data on single-family loan performance and prepare it for reporting and analytics. Several steps are required to properly prepare the data. With each step, we'll capture information like record count and elapsed time and record these in Application Insights. **Figure 5** illustrates the high-level steps in the job. We've settled on using the titles across the top of **Figure 5** to identify our separate operations.

Additionally, we've established a set of measurements with similar names (for example, Write Duration, Read Duration, Record

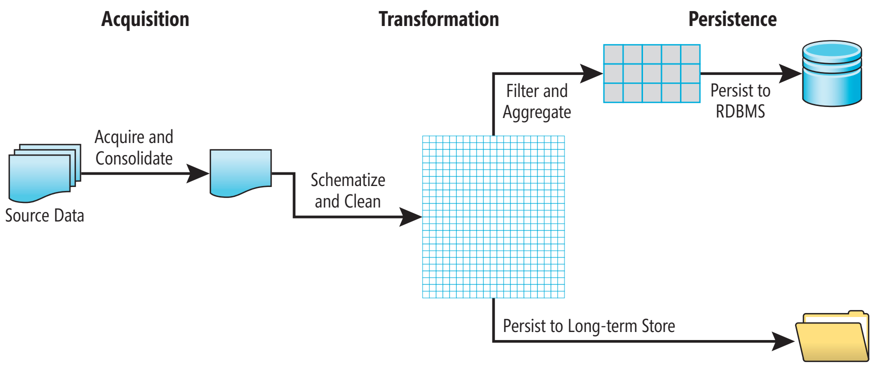


Figure 5 **Data Engineering Job Flow**

Count) that will be reported in differently named events. This will be important in the analytics as we look at specific metrics and then view them by operation or event. As shown in **Figure 5**, first we ingest multiple data files, then consolidate and transform them, and finally write to two target locations. The fully prepared data set is persisted to long-term Blob Storage and an aggregated subset is sent to our RDBMS, Azure SQL Database. Of course, within each high-level step there are several sub-steps. Specifically, we import four distinct files, merge them into a single Spark DataFrame, and write the raw, consolidated dataset to Blob Storage. The consolidated data is then read back out of Blob storage into a new DataFrame for cleansing and transformation. To complete the transformation, we subset the DataFrame (that is, narrow it to relevant columns only), rename the columns to meaningful names, and replace null values in the Servicer Name column. The final form of the data is persisted in the Parquet file format. The last step in this example persists the data to an Azure SQL Database.

For this Azure Databricks job example, we've taken the single notebook approach with the steps programmed in separate code cells. One parent operation Id is set for each run of the job. A (child) operation Id applies to each operation within the job, and we've defined Acquisition, Transformation and Persistence as these operations. We track the events occurring for each operation, recording timestamp, record count, duration and other parameters in Application Insights at job runtime.

As in the earlier predictions example, we add the Python package "applicationinsights" to the cluster, run the setup notebook, and instantiate a new instance of the TelemetryClient object. This time we'll name the instance DataEngineeringExample and then set the initial operation name to Acquisition, in order to prepare for our first series of steps to acquire source data:

```
telemetryClient = NewTelemetryClient(
  "DataEngineeringExample", operationId, parentOperationId)
telemetryClient.context.operation.name = "Acquisition"
```

Next, we capture the current time and track our first event in Application Insights, recording that the job has started:

```
import datetime

jobStartTime = datetime.datetime.now()
jobStartTimeStr = str(jobStartTime)

telemetryClient.track_event('Start Job', { 'Start Time': jobStartTimeStr,
  'perfDataFilePath':perfDataFilePath, 'perfDataFileNamePrefix' :
  perfDataFileNamePrefix, 'consolidatedDataPath':consolidatedDataPath,
  'transformedFilePath' : transformedFilePath, 'perfDBConnectionString':
  perfDBConnectionString, 'perfDBTableName': perfDBTableName})
telemetryClient.flush()
```

This is the code to set the current timestamp as the start time for the job, and record it in our first Application Insights event. First, we import the Python library datetime for convenient date and time functions, and then set variable jobStartTime to the current timestamp. It's worth noting that the signature for the track_event([eventName], [{props}], [{measurements}]) method takes parameters for the event name, dictionary of properties, and a dictionary of measurements. To that end, the timestamp variable needs to be JSON-serializable to include it in the properties of the telemetry event. So, we cast the jobStartTime object as a string and put the value in a new variable jobStartTimeStr. In the next step, we send our initial telemetry event with the track_event method, passing it our custom event name Start Time along with several parameters we selected to capture with this event. We've included parameters for various file paths and connection strings that are referenced in the job. For example, perfDataFilePath contains the location of the source data files, and perfDBConnectionString contains the connection string for the Azure SQL Database, where we'll persist some of the data. This is helpful information in such cases where we see a 0 record connect or have an alert set; we can take a quick look at the telemetry of the related operation and quickly check the files and/or databases that are being accessed.

Now we can proceed through the command cells in the notebook, adding similar event-tracking code into each step, with a few changes relevant to the inner steps of the job. Because it's often helpful to use record counts throughout a data-engineering job to consider data volume when monitoring performance and resource utilization, we've added a record count measurement to each tracked event.

Figure 6 shows a few basic data transformations, followed by event-tracking for Application Insights. Inside the exception-handling Try block, we perform three types of transformations at once on the perfTransformDF DataFrame. We subset the DataFrame, keeping only a select group of relevant columns and discarding the rest. We replace nulls in the Servicer Name column with "UNKNOWN." And, because the original column names were meaningless (for example, "_C0," "_C1"), we rename the relevant subset of columns to meaningful names like "loan_id" and "loan_age."

Once the transformations are complete, we capture the current timestamp in variable "end" as the time this step completed; count the rows in the DataFrame; and calculate the step duration based

Figure 6 **Data Transformation Event-Tracking Code**

```
if notebookError == "":
  try:
    perfTransformedDF = perfTransformedDF['_c0','_c1','_C2','_C3','_C4', \
                                          '_C5','_C6','_C7','_C8','_C9', \
                                          '_C10','_C11','_C12','_C13'] \
      .fillna({'_C2':'UNKNOWN'}) \
      .withColumnRenamed("_C0", "loan_id") \
      .withColumnRenamed("_C1", "period") \
      .withColumnRenamed("_C2", "servicer_name") \
      .withColumnRenamed("_C3", "new_int_rt") \
      .withColumnRenamed("_C4", "act_endg_upb") \
      .withColumnRenamed("_C5", "loan_age") \
      .withColumnRenamed("_C6", "mths_remng") \
      .withColumnRenamed("_C7", "aj_mths_remng") \
      .withColumnRenamed("_C8", "dt_matr") \
      .withColumnRenamed("_C9", "cd_msa") \
      .withColumnRenamed("_C10", "delq_sts") \
      .withColumnRenamed("_C11", "flag_mod") \
      .withColumnRenamed("_C12", "cd_zero_bal") \
      .withColumnRenamed("_C13", "dt_zero_bal")

    print("nulls replaced")
    end = datetime.datetime.now()

    rowCount = perfTransformedDF.count()
    duration = round((end - start).total_seconds(), 1)
    telemetryClient.track_event('Transformation Complete', {}, \
                                { 'Records Transformed': rowCount, \
                                  'Transformation Duration':duration })
    telemetryClient.flush()
  except Exception as e:
    notebookError = str(e)
    telemetryClient.track_exception(e,{"action":"column transform"},{})
else:
  print("command skipped due to previous error")
```

Yesterday's Knowledge; Tomorrow's Code!

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

DEVELOPMENT TOPICS INCLUDE:

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

REGISTER NOW

vslive.com/redmond

$400 Super Early Bird Savings Ends June 8!

Use promo code MSDN

EVENT SPONSOR Microsoft | PLATINUM SPONSOR SMARTBEAR | SILVER SPONSOR Progress | SUPPORTED BY Visual Studio, msdn magazine, Visual Studio MAGAZINE | PRODUCED BY 1105 MEDIA INC. YOUR GROWTH. OUR BUSINESS.

AGENDA AT-A-GLANCE

Visual Studio LIVE! Redmond

EXPERT SOLUTIONS FOR .NET DEVELOPERS

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

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, August 13, 2018 *(Separate entry fee required)*		
7:00 AM	8:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries		
8:00 AM	12:00 PM	**M01** Workshop: Build a Modern ASP.NET App in the Cloud with a full CI/CD Pipeline in VSTS *- Brian Randell*	**M02** Workshop: Developer Dive into SQL Server 2016 *- Leonard Lobel*	**M03** Workshop: Distributed Cross-Platform Application Architecture *- Rockford Lhotka and Jason Bock*
12:00 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center		
2:00 PM	5:30 PM	**M01** Workshop Continues	**M02** Workshop Continues	**M03** Workshop Continues
7:00 PM	9:00 PM	Dine-A-Round Dinner		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, August 14, 2018				
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:15 AM	**T01** Angular 101 *- Deborah Kurata*	**T02** Xamarin: The Future of App Development *- James Montemagno*	**T03** Cloud Oriented Programming *- Vishwas Lele*	**T04** Busting .NET Myths *- Jason Bock*	**T05** A DevOps Journey *- Abel Wang*
9:30 AM	10:45 AM	**T06** Write Object-Oriented JavaScript with TypeScript *- Rachel Appel*	**T07** Netstandard: Reuse C# Code Across Windows, Mac, Linux, iOS, Android *- Rockford Lhotka*	**T08** Microservices with ACS (Managed Kubernetes) *- Vishwas Lele*	**T09** Get Started with Git *- Robert Green*	**T10** DevOps for the SQL Server Database *- Brian Randell*
10:45 AM	11:15 AM	Sponsored Break - Visit Exhibitors - Foyer				
11:15 AM	12:15 PM	**KEYNOTE: To Be Announced**				
12:15 PM	1:30 PM	Lunch - McKinley / Visit Exhibitors - Foyer				
1:30 PM	2:45 PM	**T11** Angular Component Communication *- Deborah Kurata*	**T12** Tips & Tricks for Xamarin Development *- James Montemagno*	**T13** New SQL Server 2016 Security Features for Developers *- Leonard Lobel*	**T14** To Be Announced	**T15** Microsoft Session To Be Announced
3:00 PM	4:15 PM	**T16** Build Data Driven Web Apps Using ASP.NET Core *- Rachel Appel*	**T17** [Flutter] *- Tim Sneath*	**T18** Introduction to Azure Cosmos DB *- Leonard Lobel*	**T19** Building A Development Culture of Collaboration *- Justin Collier*	**T20** Microsoft Session To Be Announced
4:15 PM	5:45 PM	Microsoft Ask the Experts & Exhibitor Reception – Attend Exhibitor Demos				

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, August 15, 2018				
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:15 AM	**W01** Assembling the Web - A Tour of WebAssembly *- Jason Bock*	**W02** Mobile App Development for the Web Developer *- Ben Hoelting*	**W03** - SQL Server 2017 - Intelligence Built-in *- Scott Klein*	**W04** Azure DevOps with VSTS, Docker, and K8 *- Brian Randell*	**W05** Microsoft Session To Be Announced
9:30 AM	10:45 AM	**W06** Getting Pushy with SignalR and Reactive Extensions *- Jim Wooley*	**W07** Cross-Platform App Development Using Xamarin and CSLA .NET *- Rockford Lhotka*	**W08** An Architect's Guide to Data Science *- Becky Isserman*	**W09** Use Visual Studio to Scale Agile in Your Enterprise *- Richard Hundhausen*	**W10** Microsoft Session To Be Announced
11:00 AM	12:00 PM	**GENERAL SESSION: To Be Announced**				
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch - Visit Exhibitors				
1:30 PM	2:45 PM	**W11** Building Reactive Client Experiences with RxJs *- Jim Wooley*	**W12** Radically Advanced XAML: Dashboards, Timelines, Animation, and More *- Billy Hollis*	**W13** Knockout: R vs Python for Data Science *- Becky Isserman*	**W14** Develop on Cadence, Release on Demand *- Richard Hundhausen*	**W15** Microsoft Session To Be Announced
2:45 PM	3:15 PM	Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win)				
3:15 PM	4:30 PM	**W16** Building Data-Centric Single Page Apps with Angular (2,4,5...) and Breeze *- Brian Noyes*	**W17** Programming with the Model-View-ViewModel Pattern *- Miguel Castro*	**W18** Busy Developer's Guide to NoSQL *- Ted Neward*	**W19** Visualizing the Backlog with User Story Mapping *- Philip Japikse*	**W20** Data Science for Developers *- Aashish Bhateja*
6:15 PM	9:00 PM	Set Sail! VSLive's Seattle Sunset Cruise - Advanced Reservation & $20 Fee Required				

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, August 16, 2018				
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries				
8:00 AM	9:15 AM	**TH01** The Whirlwind Tour of Authentication and Authorization with ASP.NET Core *- Chris Klug*	**TH02** Getting Started Debugging C# Application *- Paul Sheriff*	**TH03** Leaders Are Made, Not Born *- Philip Japikse*	**TH04** Using The Microsoft Cognitive Custom Vision Service *- Michael Washington*	**TH05** Lessons Learned from Making Resilient Apps with Azure Mobile App Services *- Matthew Soucoup*
9:30 AM	10:45 AM	**TH06** Securing Angular Apps *- Brian Noyes*	**TH07** C# 7.x Like a Boss! *- Adam Tuliper*	**TH08** The Role of an Architect *- Ted Neward*	**TH09** Google Home Meets .NET Containers on Google Cloud *- Mete Atamel*	**TH10** Microsoft Session To Be Announced
11:00 AM	12:15 PM	**TH11** JavaScript Patterns for the C# Developer *- Ben Hoelting*	**TH12** I'll Get Back to You: Task, Await, and Asynchronous Methods *- Jeremy Clark*	**TH13** Demystifying Microservice Architecture *- Miguel Castro*	**TH14** Building Business Applications Using Bots *- Michael Washington*	**TH15** Microsoft Session To Be Announced
12:15 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center				
2:00 PM	3:15 PM	**TH16** Utilizing the MVVM Design Pattern in WPF *- Paul Sheriff*	**TH17** Getting to the Core of the .NET Standard *- Adam Tuliper*	**TH18** Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself *- Jeremy Clark*	**TH19** Wash, Rinse, Repeat: Writing Skills for Both Alexa and Cortana *- Christine Flora*	**TH20** Microsoft Session To Be Announced
3:30 PM	4:45 PM	**TH21** WPF Styles, Resources and Templates, Oh My! *- Paul Sheriff*	**TH22** Building Apps with Microsoft Graph and Visual Studio *- Robert Green*	**TH23** How to Interview a Developer *- Billy Hollis*	**TH24** The Complete Package: Creating a Deployable Solution for Microsoft Teams *- Christine Flora*	**TH25** Microsoft Session To Be Announced

START TIME	END TIME	Visual Studio Live! Post-Conference Workshops: Friday, August 17, 2018 *(Separate entry fee required)*	
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries	
8:00 AM	5:00 PM	**F01** Workshop: UX Design for Developers *- Billy Hollis*	**F02** Workshop: Web Developerment in 2018 *- Chris Klug*

Speakers and sessions subject to change

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

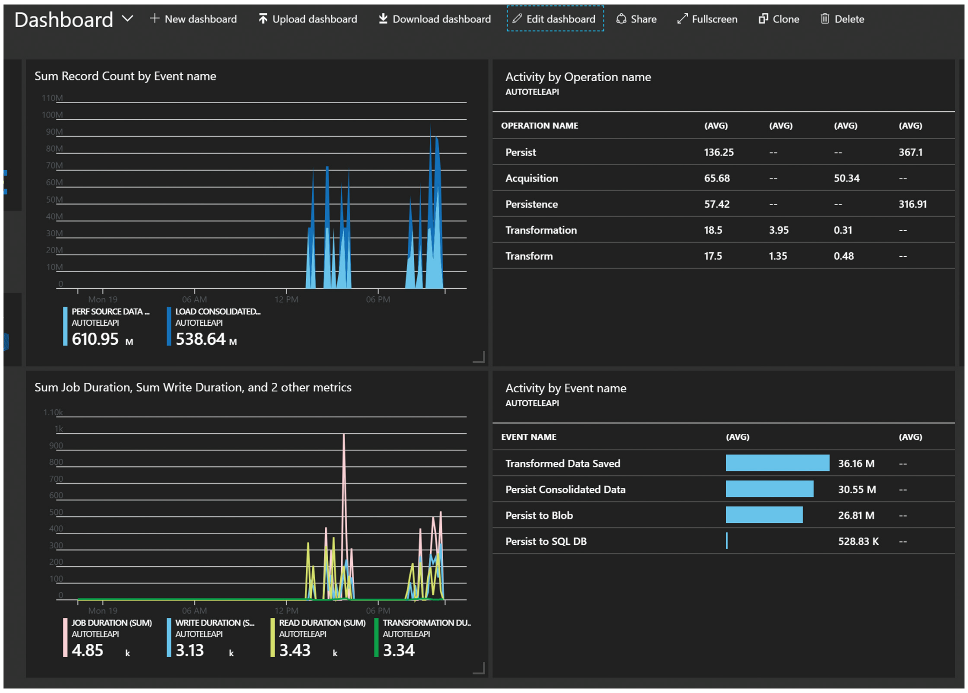
linkedin.com – Join the "Visual Studio Live" group!

vslive.com/redmond

on start and end times. We send that telemetry to Application Insights with the telemetryClient.track_event method using the event name "Transformation Complete," and we include measurements for records transformed and transformation duration.

We add some very basic exception handling into our notebooks purely to illustrate tracking exceptions with Application Insights, as well. Note within the except block in **Figure 6** that if we catch an exception, we're going to call the track_exception method. We pass the exception as the first parameter and the subsequent parameters are the same types as in track_event, allowing you to record as much information around the event as possible. One important note to make here is that there's currently no exception-handling semantics for inline sql. So, it might be best to skip magics like %sql for production jobs until support for exception handling is added.

The other steps in our data-engineering job, including operations for Acquisition and Persistence, follow the pattern seen in the Transformation code for sending telemetry events with custom measurements to Application Insights.



Dashboard
New dashboard
Upload dashboard
Download dashboard
Edit dashboard
Share
Fullscreen
Clone
Delete
Sum Record Count by Event name
110M
100M
90M
80M
70M
60M
50M
40M
30M
20M
10M
0
Mon 19
06 AM
12 PM
06 PM
PERF SOURCE DATA ... AUTOTELEAPI 610.95 M
LOAD CONSOLIDATED... AUTOTELEAPI 538.64 M
Sum Job Duration, Sum Write Duration, and 2 other metrics
1.10k
1k
900
800
700
600
500
400
300
200
100
0
Mon 19
06 AM
12 PM
06 PM
JOB DURATION (SUM) AUTOTELEAPI 4.85 k
WRITE DURATION (S... AUTOTELEAPI 3.13 k
READ DURATION (SUM) AUTOTELEAPI 3.43 k
TRANSFORMATION DU... AUTOTELEAPI 3.34

Activity by Operation name
AUTOTELEAPI

OPERATION NAME	(AVG)	(AVG)	(AVG)	(AVG)
Persist	136.25	--	--	367.1
Acquisition	65.68	--	50.34	--
Persistence	57.42	--	--	316.91
Transformation	18.5	3.95	0.31	--
Transform	17.5	1.35	0.48	--

Activity by Event name
AUTOTELEAPI

EVENT NAME	(AVG)	(AVG)
Transformed Data Saved	36.16 M	--
Persist Consolidated Data	30.55 M	--
Persist to Blob	26.81 M	--
Persist to SQL DB	528.83 K	--

Figure 7 **Application Insights Charts on Azure Dashboard**

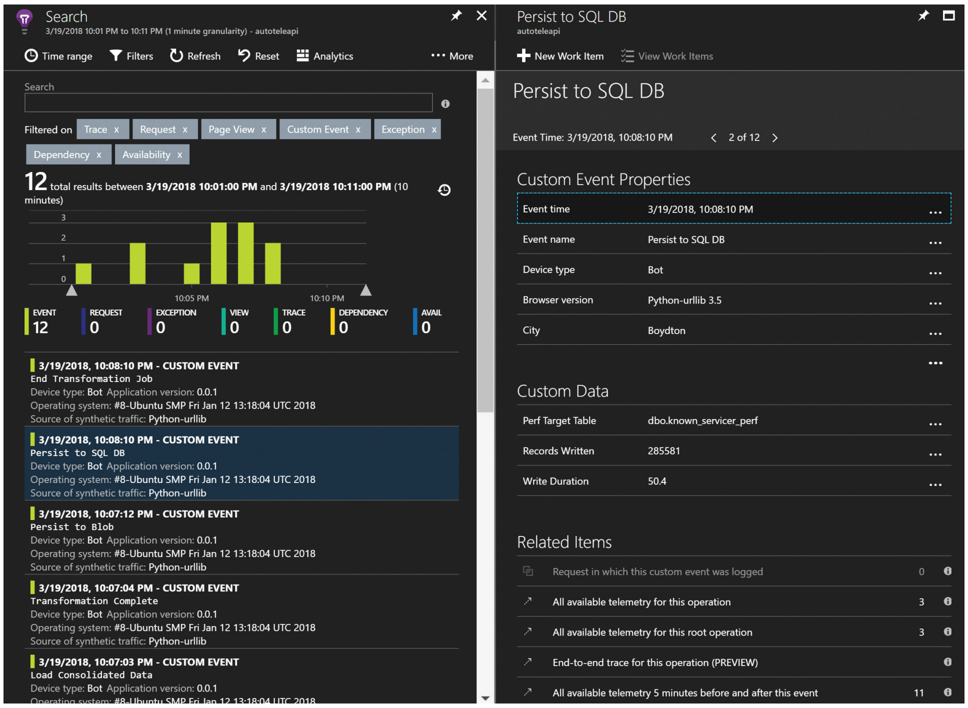
Configuring Analytics and Alerts

With the code in place to send the telemetry, we turn to configuring Application Insights to create live dashboards, look through event and correlated event details, and set up alerts to inform and possibly take action based on the event trigger.

Figure 7 depicts a few charts that we've configured via the Metrics Explorer blade and the Metrics (preview) blade in Application Insights and then pinned to our Azure Portal Dashboard.

Take note of the right two quartiles. The top right shows a grid of durations grouped by the operation name that we reported the telemetry under when we added the tracking calls. The bottom right shows a record count measurement grouped by the event name we used. Sure enough, "Persist to SQL DB" is much lower than the others, as this was the event that wrote only a small, filtered subset of our data to Azure SQL Database. Choosing your operation groupings, operation names, and event names is an important part of planning that pays off at this point as you get to visualize and report on the data in a way that makes sense for how you think about your operations.

The left two quartiles in **Figure 7** show charts that were created with the Metrics (preview), which has a nice configuration UI, as well as some additional functionality for splitting the measurements based on another property. In the top left



3/19/2018 10:01 PM to 10:11 PM (1 minute granularity) - autoteleapi

12 total results between **3/19/2018 10:01:00 PM** and **3/19/2018 10:11:00 PM** (10 minutes)

3/19/2018, 10:08:10 PM - CUSTOM EVENT
`End Transformation Job`
Device type: Bot Application version: 0.0.1
Operating system: #8-Ubuntu SMP Fri Jan 12 13:18:04 UTC 2018
Source of synthetic traffic: Python-urllib

3/19/2018, 10:08:10 PM - CUSTOM EVENT
`Persist to SQL DB`
Device type: Bot Application version: 0.0.1
Operating system: #8-Ubuntu SMP Fri Jan 12 13:18:04 UTC 2018
Source of synthetic traffic: Python-urllib

3/19/2018, 10:07:12 PM - CUSTOM EVENT
`Persist to Blob`
Device type: Bot Application version: 0.0.1
Operating system: #8-Ubuntu SMP Fri Jan 12 13:18:04 UTC 2018
Source of synthetic traffic: Python-urllib

3/19/2018, 10:07:04 PM - CUSTOM EVENT
`Transformation Complete`
Device type: Bot Application version: 0.0.1
Operating system: #8-Ubuntu SMP Fri Jan 12 13:18:04 UTC 2018
Source of synthetic traffic: Python-urllib

3/19/2018, 10:07:03 PM - CUSTOM EVENT
`Load Consolidated Data`
Device type: Bot Application version: 0.0.1
Operating system: #8-Ubuntu SMP Fri Jan 12 13:18:04 UTC 2018

Persist to SQL DB
autoteleapi

Persist to SQL DB

Event Time: 3/19/2018, 10:08:10 PM   < 2 of 12 >

Custom Event Properties

Event time	3/19/2018, 10:08:10 PM
Event name	Persist to SQL DB
Device type	Bot
Browser version	Python-urllib 3.5
City	Boydton

Custom Data

Perf Target Table	dbo.known_servicer_perf
Records Written	285581
Write Duration	50.4

Related Items

Request in which this custom event was logged	0
All available telemetry for this operation	3
All available telemetry for this root operation	3
End-to-end trace for this operation (PREVIEW)	
All available telemetry 5 minutes before and after this event	11

Figure 8 **Event Search and Details**

you can see the record count, but we've split it so that this is reported by Event Name, giving us a graph and data for the record count for different events. Here we're comparing record counts taken when the source data was read to record counts taken later when consolidated data was loaded into a DataFrame. This is an important feature since Record Count might be a pretty common measurement across our parent operation, but we'd like to see it at each operation or event.

If you see something in one of the operational graphs that calls for some research, you can search through all the telemetry. **Figure 8** depicts the results of a search and a graph showing an occurrence count over time in the left pane. In the right pane you can look at all of the information recorded in the event. Recall the track_event([name], [properties], [measurements]) signature. We've pulled up the detail of a Persist to SQL DB event in which you can see the custom properties at the top. In the middle, labeled Custom Data, is where you can find the custom measurements that were sent with the telemetry. At the bottom right are all of the Related Items where you can easily navigate to all of the events that belong to the operation or parent operation. In addition, there's a line at the bottom to see all available telemetry at the time of the event. If you've standardized on Application Insights for your runtime monitoring, this is a great tool for understanding the overall system state and the operational context of an event. Having insight into what's going on broadly might help explain when record counts are off or duration is askew.

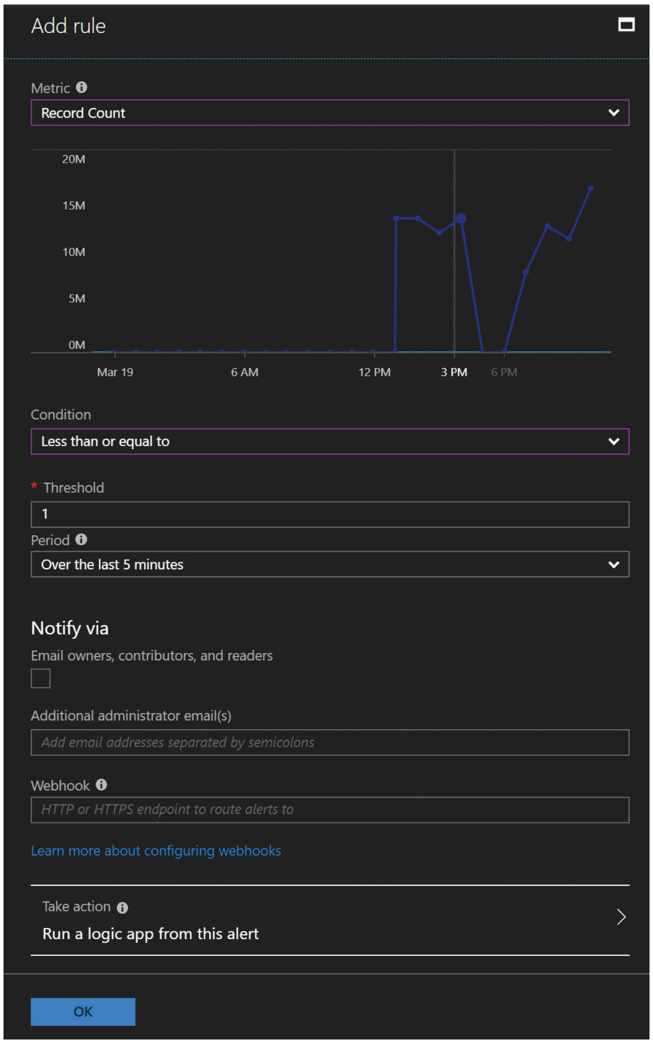
The last thing we want to cover for Application Insights is the ability to set up an alert. In **Figure 9** you can see part of the alert configuration. Like the other elements we looked at, the custom information we sent in the events shows up here for us to choose as criteria for alerting.

Having insight into what's going on broadly might help explain when record counts are off or duration is askew.

As you might expect, the alert can send an e-mail. However, it can also call a WebHook, which makes for a nice and easy way to take any other action you might desire. Azure Functions is a perfect fit for this setup and will allow you to create whatever custom action you like. More interestingly, Application Insights is directly integrated with Logic Apps. This enables native capability to integrate and orchestrate actions across a wide variety of integrations and Microsoft Azure. Thus, an Application Insights alert could notify people while starting to take compensating and/or corrective actions via Logic Apps orchestration, including actions and integrations with downstream and upstream systems.

In Closing

We want to make sure we highlight the key bits of information. Application Insights is not a Log Analytics solution. It integrates with Azure Log Analytics, which provides post hoc analysis and long-term log retention. Application Insights is for monitoring and analytics of your runtime operations, giving you information, insights and alerts about what's happening now. Direct integration with other Azure services and broad availability of platform SDKs makes it a nice fit to help operationalize your Azure Databricks jobs. As a result, monitoring for those jobs isn't done in a silo, but rather within the context of the full solution architecture. ■


Add rule

Metric: Record Count

Condition: Less than or equal to

* Threshold: 1

Period: Over the last 5 minutes

Notify via

Email owners, contributors, and readers

Additional administrator email(s): Add email addresses separated by semicolons

Webhook: HTTP or HTTPS endpoint to route alerts to

Learn more about configuring webhooks

Take action: Run a logic app from this alert

OK

Figure 9 **Setting Up an Alert on Metric**

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

Ryan Murphy *is a solution architect living in Saint Louis, Mo. He's been building and innovating with data for nearly 20 years, including extensive work in the gaming and agriculture industries. Currently, Murphy is helping some of the world's largest organizations modernize their business with data solutions powered by Microsoft Azure Cloud. Follow him on Twitter: @murphrp.*

Thanks to the following Microsoft technical expert for reviewing this article: *Darren Brust*

Effective Async with Coroutines and C++/WinRT

Kenny Kerr

The Windows Runtime has a relatively simple async model in the sense that, like everything else in the Windows Runtime, it's focused on allowing components to expose async methods and making it simple for apps to call those async methods. It doesn't in itself provide a concurrency runtime or even anything in the way of building blocks for producing or consuming async methods. Instead, all of that is left to the individual language projections. This is as it should be and isn't meant to trivialize the Windows Runtime async pattern. It's no small feat to implement this pattern correctly. Of course, it also means that a developer's perception of async in the Windows Runtime is very heavily influenced by the developer's language of choice. A developer who has only ever used C++/CX might, for example, wrongly but understandably assume that async is a hot mess.

The ideal concurrency framework for the C# developer will be different from the ideal concurrency library for the C++ developer. The role of the language, and libraries in the case of C++, is to take care of the mechanics of the async pattern and provide a natural bridge to a language-specific implementation.

Coroutines are the preferred abstraction for both implementing and calling async methods in C++, but let's first make sure we understand how the async model works. Consider a class with a single static method that looks something like this:

```
struct Sample
{
  Sample() = delete;

  static Windows::Foundation::IAsyncAction CopyAsync();
};
```

Async methods end with "Async" by convention, so you might think of this as the Copy async method. There might be a blocking or synchronous alternative that's simply called Copy. It's conceivable that a caller might want a blocking Copy method for use by a background thread and a non-blocking, or asynchronous, method for use by a UI thread that can't afford to block for fear of appearing unresponsive.

At first, the CopyAsync method might seem quite simple to call. I could write the following C++ code:

```
IAsyncAction async = Sample::CopyAsync();
```

As you might imagine, the resulting IAsyncAction isn't actually the ultimate result of the async method, even though it's the result of calling the CopyAsync method in a traditional procedural manner. The IAsyncAction is the object that a caller might use to wait upon the result synchronously or asynchronously, depending on the situation. Along with IAsyncAction, there are three other

This article discusses:

- Concurrency
- Coroutines
- The thread pool

Technologies discussed:

Windows Runtime, ISO C++17, C++ Coroutines, C++/WinRT

Figure 1 **A Comparison of Async Interfaces**

Name	Result	Progress
IAsyncAction	No	No
IAsyncActionWithProgress	No	Yes
IAsyncOperation	Yes	No
IAsyncOperationWithProgress	Yes	Yes

well-known interfaces that follow a similar pattern and offer different features for the callee to communicate information back to the caller. The table in **Figure 1** provides a comparison of the four async interfaces.

In C++ terms, the interfaces can be expressed as shown in **Figure 2**.

IAsyncAction and IAsyncActionWithProgress can be waited upon to determine when the async method completes, but these interfaces don't offer any observable result or return value directly. IAsyncOperation and IAsyncOperationWithProgress, on the other hand, expect the Result type parameter to indicate the type of result that can be expected when the async method completes successfully. Finally, IAsyncActionWithProgress and IAsyncOperationWithProgress expect the Progress type parameter to indicate the type of progress information that can be expected periodically for long-running operations up until the async method completes.

The role of the language, and libraries in the case of C++, is to take care of the mechanics of the async pattern and provide a natural bridge to a language-specific implementation.

There are a few ways to wait upon the result of an async method. I won't describe them all here as that would turn this into a very long article. While there are a variety of ways to handle async completion, there are only two that I recommend: the async.get method, which performs a blocking wait, and the co_await async expression, which performs a cooperative wait in the context of a coroutine. Neither is better than the other as they simply serve different purposes. Let's look now at how to do a blocking wait.

As I mentioned, a blocking wait can be achieved using the get method as follows:

```
IAsyncAction async = Sample::CopyAsync();
async.get();
```

There's seldom any value in holding on to the async object and the following form is thus preferred:

```
Sample::CopyAsync().get();
```

It's important to keep in mind that the get method will block the calling thread until the async method completes. As such, it's not appropriate to use the get method on a UI thread because it may cause the app to become unresponsive. An assertion will fire in unoptimized builds if you attempt to do so. The get method is ideal for console apps or background threads where, for whatever reason, you may not want to use a coroutine.

Once the async method completes, the get method will return any result directly to the caller. In the case of IAsyncAction and IAsyncActionWithProgress, the return type is void. That might be useful for an async method that initiates a file-copy operation, but less so for something like an async method that reads the contents of a file. Let's add another async method to the example:

```
struct Sample
{
  Sample() = delete;

  static Windows::Foundation::IAsyncAction CopyAsync();
  static Windows::Foundation::IAsyncOperation<hstring> ReadAsync();
};
```

In the case of ReadAsync, the get method will properly forward the hstring result to the caller once the operation completes:

```
Sample::CopyAsync().get();

hstring result = Sample::ReadAsync().get();
```

Assuming execution returns from the get method, the resulting string will hold whatever value was returned by the async method upon its successful completion. Execution may not return, for example, if an error occurred.

The get method is limited in the sense that it can't be used from a UI thread, nor does it exploit the full potential of the machine's concurrency, since it holds the calling thread hostage until the async method completes. Using a coroutine allows the async method to complete without holding such a precious resource captive for some indeterminate amount of time.

Handling Async Completion

Now that you have a handle on async interfaces in general, let's begin to drill down into how they work in a bit more detail. Assuming you're not satisfied with the blocking wait provided by the get method, what other options are there? We'll soon switch gears and focus entirely on coroutines but, for the moment, let's take a closer look at those async interfaces to see what they offer. Both the coroutine support, as well as the get method I looked at previously, rely on the contract and state machine implied by those interfaces. I won't go into too much detail because you really don't need to know all that much about these, but let's explore the basics so they'll at least be familiar if you do ever have to dive in and use them directly for something more out of the ordinary.

Figure 2 **The Async Interfaces Expressed in C++ Terms**

```
namespace Windows::Foundation
{
  struct IAsyncAction;

  template <typename Progress>
  struct IAsyncActionWithProgress;

  template <typename Result>
  struct IAsyncOperation;

  template <typename Result, typename Progress>
  struct IAsyncOperationWithProgress;
}
```

All four of the async interfaces logically derive from the IAsyncInfo interface. There's very little you can do with IAsyncInfo and it's regrettable that it even exists because it adds a bit of overhead. The only IAsyncInfo members you should really consider are Status, which can tell you whether the async method has completed, and Cancel, which can be used to request cancellation of a long-running operation whose result is no longer needed. I nitpick this design because I really like the async pattern in general and just wish it were perfect because it's so very close.

The Status member can be useful if you need to determine whether an async method has completed without actually waiting for it. Here's an example:

```
auto async = ReadAsync();

if (async.Status() == AsyncStatus::Completed)
{
  auto result = async.GetResults();
  printf("%ls\n", result.c_str());
}
```

Each of the four async interfaces, not IAsyncInfo itself, provides individual versions of the GetResults method that should be called only after you've determined that the async method has completed.

Figure 3 **Notifying a Waiting Thread Using an Event**

```
template <typename T>
auto get(T const& async)
{
  if (async.Status() != AsyncStatus::Completed)
  {
    handle signal = CreateEvent(nullptr, true, false, nullptr);

    async.Completed([&](auto&&, auto&&)
    {
      SetEvent(signal.get());
    });

    WaitForSingleObject(signal.get(), INFINITE);
  }

  return async.GetResults();
}
```

Figure 4 **Using a Condition Variable with a Slim Reader/Writer Lock**

```
template <typename T>
auto get(T const& async)
{
  if (async.Status() != AsyncStatus::Completed)
  {
    slim_mutex m;
    slim_condition_variable cv;
    bool completed = false;

    async.Completed([&](auto&&, auto&&)
    {
      {
        slim_lock_guard const guard(m);
        completed = true;
      }

      cv.notify_one();
    });

    slim_lock_guard guard(m);
    cv.wait(m, [&] { return completed; });
  }

  return async.GetResults();
}
```

Don't confuse this with the get method provided by C++/WinRT. While GetResults is implemented by the async method itself, get is implemented by C++/WinRT. GetResults won't block if the async method is still running and will likely throw an hresult_illegal_method_call exception if called prematurely. You can, no doubt, begin to imagine how the blocking get method is implemented. Conceptually, it looks something like this:

```
auto get() const
{
  if (Status() != AsyncStatus::Completed)
  {
    // Wait for completion somehow ...
  }

  return GetResults();
}
```

The actual implementation is a bit more complicated, but this captures the gist of it. The point here is that GetResults is called regardless of whether it's an IAsyncOperation, which returns a value, or IAsyncAction, which doesn't. The reason for this is that GetResults is responsible for propagating any error that may have occurred within the implementation of the async method and will rethrow an exception as needed.

The question that remains is how the caller can wait for completion. I'm going to write a non-member get function to show you what's involved. I'll start with this basic outline, inspired by the previous conceptual get method:

```
template <typename T>
auto get(T const& async)
{
  if (async.Status() != AsyncStatus::Completed)
  {
    // Wait for completion somehow ...
  }

  return async.GetResults();
}
```

I want this function template to work with all four of the async interfaces, so I'll use the return statement unilaterally. Special provision is made in the C++ language for genericity and you can be thankful for that.

Each of the four async interfaces provides a unique Completed member that can be used to register a callback—called a delegate—that will be called when the async method completes. In most cases, C++/WinRT will automatically create the delegate for you. All you need to do is provide some function-like handler, and a lambda is usually the simplest:

```
async.Completed([](auto&& async, AsyncStatus status)
{
  // It's done!
});
```

The type of the delegate's first parameter will be that of the async interface that just completed, but keep in mind that completion should be regarded as a simple signal. In other words, don't stuff a bunch of code inside the Completed handler. Essentially, you should regard it as a noexcept handler because the async method won't itself know what to do with any failure occurring inside this handler. So what can you do?

Well, you could simply notify a waiting thread using an event. **Figure 3** shows what the get function might look like.

C++/WinRT's get methods use a condition variable with a slim reader/writer lock because it's slightly more efficient. Such a variant might look something like what's shown in **Figure 4**.

You can, of course, use the C++ standard library's mutex and condition variable if you prefer. The point here is simply that the Completed handler is your hook to wiring up async completion and it can be done quite generically.

Naturally, there's no reason for you to write your own get function, and more than likely coroutines will be much simpler and more versatile in general. Still, I hope this helps you appreciate some of the power and flexibility in the Windows Runtime.

Producing Async Objects

Now that we've explored the async interfaces and some completion mechanics in general, let's turn our attention to creating or producing implementations of those four async interfaces. As you've already learned, implementing WinRT interfaces with C++/WinRT is very simple. I might, for example, implement IAsyncAction, as shown in **Figure 5**.

The difficulty comes when you consider how you might implement those methods. While it's not hard to imagine some implementation, it's almost impossible to do this correctly without first reverse-engineering how the existing language projections actually implement them. You see, the WinRT async pattern only works if everyone implements these interfaces using a very specific state machine and in exactly the same way. Each language projection makes the same assumptions about how this state machine is implemented and if you happen to implement it in a slightly different way, bad things will happen.

Thankfully, you don't have to worry about this because each language projection, with the exception of C++/CX, already implements this correctly for you. Here's a complete implementation of IAsyncAction thanks to C++/WinRT's coroutine support:

```
IAsyncAction CopyAsync()
{
  co_return;
}
```

Now this isn't a particularly interesting implementation, but it is very educational and a good example of just how much C++/WinRT is doing for you. Because this is a complete implementation, we can use it to exercise some of what we've learned thus far. The earlier CopyAsync function is a coroutine. The coroutine's return type is used to stitch together an implementation of both IAsyncAction and IAsyncInfo, and the C++ compiler brings it to life at just the right moment. We'll explore some of those details later, but for now let's observe how this coroutine works. Consider the following console app:

```
IAsyncAction CopyAsync()
{
  co_return;
}

int main()
{
  IAsyncAction async = CopyAsync();

  async.get();
}
```

The main function calls the CopyAsync function, which returns an IasyncAction. If you forget for a moment what the CopyAsync function's body or definition looks like, it should be evident that it's just a function that returns an IAsyncAction object. You can therefore use it in all the ways that you've already learned.

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

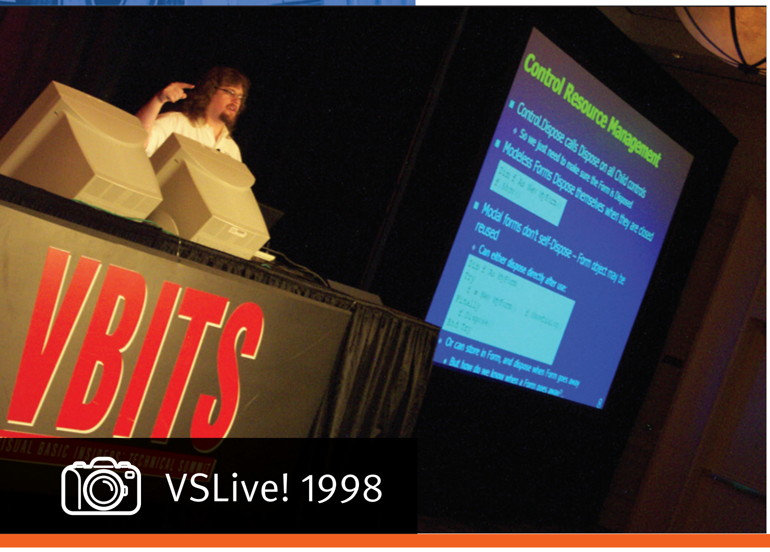
September 17-20, 2018
Renaissance
Chicago

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

Look Back to Code Forward

Visual Studio Live! (VSLive!™) is thrilled to be returning to Chicago where developers, software architects, engineers and designers will "look back to code forward" during four days of unbiased and cutting-edge education on the Microsoft Platform.

Tackle training on the hottest topics (like .NET Core, Angular, VS2017), debate with industry and Microsoft insiders (people like Rockford Lhotka, Deborah Kurata and Brock Allen) and network with your peers—plus, help us celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Come experience the education, knowledge-share and networking at #VSLive25.



VSLive! 1998

VSLive! 2016

SILVER SPONSOR
JET BRAINS

SUPPORTED BY
Microsoft
Visual Studio
msdn magazine
Visual Studio MAGAZINE

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

DEVELOPMENT TOPICS INCLUDE:

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

WHAT #VSLIVE25 CAN DO FOR YOU!

"I originally came to Visual Studio Live! to improve my technical skills, but I came back in order to network and meet other developers. The biggest change I've made at work, by far, is the quality of unit tests, which has resulted in a significant drop in reported bugs."

– Justin C Kritzer, Lehigh County Government Center

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

– Abby Quick, Epiq

Register Now to Save $400!

Use Promo Code MSDN

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/chicago

Figure 5 **Implementing IAsyncAction**

```
struct MyAsync : implements<MyAsync, IAsyncAction, IAsyncInfo>
{
  // IAsyncInfo members ...
  uint32_t Id() const;
  AsyncStatus Status() const;
  HRESULT ErrorCode() const;
  void Cancel() const;
  void Close() const;

  // IAsyncAction members ...
  void Completed(AsyncActionCompletedHandler const& handler) const;
  AsyncActionCompletedHandler Completed() const;
  void GetResults() const;
};
```

Figure 6 **A Function That Downloads and Returns a Cached Value**

```
hstring m_cache;

IAsyncOperation<hstring> ReadAsync()
{
  if (m_cache.empty())
  {
    // Download and cache value ...
  }

  co_return m_cache;
}

int main()
{
  hstring message = ReadAsync().get();
  printf("%ls\n", message.c_str());
}
```

A coroutine (of this sort) must have a co_return statement or a co_await statement. It may, of course, have multiple such statements, but must have at least one of these in order to actually be a coroutine. As you might expect, a co_return statement doesn't introduce any kind of suspension or asynchrony. Therefore, this CopyAsync function produces an IAsyncAction that completes immediately or synchronously. I can illustrate this as follows:

```
IAsyncAction Async()
{
  co_return;
}

int main()
{
  IAsyncAction async = Async();
  assert(async.Status() == AsyncStatus::Completed);
}
```

The assertion is guaranteed to be true. There's no race here. Because CopyAsync is just a function, the caller is blocked until it returns and the first opportunity for it to return happens to be the co_return statement. What this means is that if you have some async contract that you need to implement, but the implementation doesn't actually need to introduce any asynchrony, it can simply return the value directly, and without blocking or introducing a context switch. Consider a function that downloads and then returns a cached value, as shown in **Figure 6**.

The first time ReadAsync is called, the cache is likely empty and the result is downloaded. Presumably this will suspend the coroutine itself while this takes place. Suspension implies that execution returns to the caller. The caller is handed an async object that has not, in fact, completed, hence the need to somehow wait for completion.

The beauty of coroutines is that there's a single abstraction both for producing async objects and for consuming those same async objects. An API or component author might implement an async method as described earlier, but an API consumer or app developer could also use coroutines to call and wait for its completion. Let's now rewrite the main function from **Figure 6** to use a coroutine to do the waiting:

```
IAsyncAction MainAsync()
{
  hstring result = co_await ReadAsync();
  printf("%ls\n", result.c_str());
}

int main()
{
  MainAsync().get();
}
```

I have essentially taken the body of the old main function and moved it into the MainAsync coroutine. The main function uses the get method to prevent the app from terminating while the coroutine completes asynchronously. The MainAsync function has something new and that's the co_await statement. Rather than using the get method to block the calling thread until ReadAsync completes, the co_await statement is used to wait for the ReadAsync function to complete in a cooperative or non-blocking manner. This is what I meant by a suspension point. The co_await statement represents a suspension point. This app only calls ReadAsync once, but you can imagine it being called multiple times in a more interesting app. The first time it gets called, the MainAsync coroutine will actually suspend and return control to its caller. The second time it's called, it won't suspend at all but rather return the value directly.

Coroutines are very new to many C++ developers, so don't feel bad if this still seems rather magical. These concepts will become quite clear as you start writing and debugging coroutines yourself. The good news is that you already know enough to begin to make effective use of coroutines to consume async APIs provided by Windows. For example, you should be able to reason about how the console app in **Figure 7** works.

Figure 7 **An Example Console App with Coroutines**

```
#include "winrt/Windows.Web.Syndication.h"

using namespace winrt;
using namespace Windows::Foundation;
using namespace Windows::Web::Syndication;

IAsyncAction MainAsync()
{
  Uri uri(L"https://kennykerr.ca/feed");
  SyndicationClient client;
  SyndicationFeed feed = co_await client.RetrieveFeedAsync(uri);

  for (auto&& item : feed.Items())
  {
    hstring title = item.Title().Text();

    printf("%ls\n", title.c_str());
  }
}

int main()
{
  init_apartment();
  MainAsync().get();
}
```

Give it a try right now and see just how much fun it is to use modern C++ on Windows.

Coroutines and the Thread Pool

Creating a basic coroutine is trivial. You can very easily co_await some other async action or operation, simply co_return a value, or craft some combination of the two. Here's a coroutine that's not asynchronous at all:

```
IAsyncOperation<int> return_123()
{
  co_return 123;
}
```

Even though it executes synchronously, it still produces a completely valid implementation of the IAsyncOperation interface:

```
int main()
{
  int result = return_123().get();
  assert(result == 123);
}
```

Here's one that will wait for five seconds before returning the value:

```
using namespace std::chrono;

IAsyncOperation<int> return_123_after_5s()
{
  co_await 5s;
  co_return 123;
}
```

The next one is ostensibly going to execute asynchronously and yet the main function remains largely unchanged, thanks to the get function's blocking behavior:

```
int main()
{
  int result = return_123_after_5s().get();
  assert(result == 123);
}
```

The co_return statement in the last coroutine will execute on the Windows thread pool, because the co_await expression is a chrono duration that uses a thread pool timer. The co_await statement represents a suspension point and it should be apparent that a coroutine may resume on a completely different thread following suspension. You can also make this explicit using resume_background:

```
IAsyncOperation<int> background_123()
{
  co_await resume_background();
  co_return 123;
}
```

There's no apparent delay this time, but the coroutine is guaranteed to resume on the thread pool. What if you're not sure? You might have a cached value and only want to introduce a context switch if the value must be retrieved from latent storage. This is where it's good to remember that a coroutine is also a function, so all the normal rules apply:

```
IAsyncOperation<int> background_123()
{
  static std::atomic<int> result{0};

  if (result == 0)
  {
    co_await resume_background();
    result = 123;
  }

  co_return result;
}
```

This is only conditionally going to introduce concurrency. Multiple threads could conceivably race in and call background_123, causing a few of them to resume on the thread pool, but eventually the atomic variable will be primed and the coroutine will begin to complete synchronously. That is, of course, the worst case.

Let's imagine the value may only be read from storage once a signal is raised, indicating that the value is ready. We can use the two coroutines in **Figure 8** to pull this off.

Figure 8 **Reading a Value from Storage After a Signal Is Raised**

```
handle m_signal{ CreateEvent(nullptr, true, false, nullptr) };
std::atomic<int> m_value{ 0 };

IAsyncAction prepare_result()
{
  co_await 5s;
  m_value = 123;
  SetEvent(m_signal.get());
}

IAsyncOperation<int> return_on_signal()
{
  co_await resume_on_signal(m_signal.get());
  co_return m_value;
}
```

The first coroutine artificially waits for five seconds, sets the value, and then signals the Win32 event. The second coroutine waits for the event to become signaled, and then simply returns the value. Once again, the thread pool is used to wait for the event, leading to an efficient and scalable implementation. Coordinating the two coroutines is straightforward:

```
int main()
{
  prepare_result();

  int result = return_on_signal().get();
  assert(result == 123);
}
```

The main function kicks off the first coroutine but doesn't block waiting for its completion. The second coroutine immediately begins waiting for the value, blocking as it does so.

Wrapping Up

Well, I've gone pretty deep on async and coroutines in C++. In this article I've focused mostly on the thread pool, or what might be called background threads, but you can dive deeper by visiting the online version of this article at `aka.ms/M1h2v0`. The Web article includes an additional section that explores ways to take precise control over execution context. Coroutines can be used to introduce concurrency or deal with latency in other APIs, so it helps to address confusion that can arise around the execution context of a given coroutine at any particular point in time. Check out the online-exclusive content in the article at `aka.ms/M1h2v0` to learn more. There's always more to say about concurrency. It's such a fascinating topic, and I hope this introduction will get you excited about how simple it is to deal with async in your C++ apps and components and the sheer power at your fingertips when you begin to use C++/WinRT. ■

Kenny Kerr *is an author, systems programmer and the creator of C++/WinRT. He's also an engineer on the Windows team at Microsoft where he's designing the future of C++ for Windows, enabling developers to write beautiful highperformance apps and components with incredible ease.*

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

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

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

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

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

REGISTER BY 6/1 TO **SAVE $400!**

Use Promo Code MSDN

EVENT PARTNER Microsoft

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

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

AGENDA AT-A-GLANCE

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

START TIME	END TIME				
START TIME	**END TIME**	**TechMentor Pre-Conference Workshops: Monday, August 6, 2018** *(Separate entry fee required)*			
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Light Breakfast			
9:00 AM	12:00 PM	**M01** Workshop: How to Prevent all Ransomware / Malware in 2018 - *Sami Laiho*	**M02** Workshop: Building Office 365 Federated Identity from Scratch Using AD FS - *Nestori Syynimaa*	**M03** Workshop: Managing Windows Server with Project Honolulu - *Dave Kawula*	
12:00 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
2:00 PM	5:00 PM	**M01** Workshop: How to Prevent all Ransomware / Malware in 2018 (Continues) - *Sami Laiho*	**M04** Workshop: Master PowerShell Tricks for Windows Server 2016 and Windows 10 - *Will Anderson & Thomas Rayner*	**M05** Workshop: Dave Kawula's Notes from the Field on Microsoft Storage Spaces Direct - *Dave Kawula*	
6:30 PM	8:30 PM	Dine-A-Round Dinner - Suite in Hyatt Regency Lobby			
START TIME	**END TIME**	**TechMentor Day 1: Tuesday, August 7, 2018**			
7:00 AM	8:00 AM	Registration - Coffee and Light Breakfast			
8:00 AM	9:15 AM	**T01** Enterprise Client Management in a Modern World - *Kent Agerlund*	**T02** How to Write (PowerShell) Code that Doesn't Suck - *Thomas Rayner*	**T03** The Easy Peasy of Troubleshooting Azure - *Mike Nelson*	**T04** Nine O365 Security Issues Microsoft Probably Hasn't Told You (and You Probably Don't Want to Know) - *Nestori Syynimaa*
9:30 AM	10:45 AM	**T05** Managing Client Health—Getting Close to the Famous 100% - *Kent Agerlund*	**T06** The Network is Slow! Or is it? Network Troubleshooting for Windows Administrators - *Richard Hicks*	**T07** Getting Started with PowerShell 6.0 for IT Pro's - *Sven van Rijen*	**T08** The Weakest Link of Office 365 Security - *Nestori Syynimaa*
11:00 AM	12:00 PM	**KEYNOTE: To Be Announced** - *Stephen L. Rose, Sr. PMM, One Drive For Business, Microsoft*			
12:00 PM	1:00 PM	Lunch - *McKinley* / Visit Exhibitors - *Foyer*			
1:00 PM	2:15 PM	**T09** How to Get Started with Microsoft EMS Right Now - *Peter Daalmans*	**T10** Back to the Future! Access Anywhere with Windows 10 Always on VPN - *Richard Hicks*	**T11** Using Desired State Configuration in Azure - *Will Anderson*	**T12** To Be Announced
2:15 PM	2:45 PM	Sponsored Break - Visit Exhibitors - *Foyer*			
2:45 PM	4:00 PM	**T13** Conceptualizing Azure Resource Manager Templates - *Will Anderson*	**T14** How to Use PowerShell to Become a Windows Management SuperHero - *Petri Paavola*	**T15** Making the Most Out of the Azure Dev/Test Labs - *Mike Nelson*	**T16** To Be Announced
4:00 PM	5:30 PM	Exhibitor Reception – Attend Exhibitor Demo - *Foyer*			
START TIME	**END TIME**	**TechMentor Day 2: Wednesday, August 8, 2018**			
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast			
8:00 AM	9:15 AM	**W01** Automated Troubleshooting Techniques in Enterprise Domains (Part 1) - *Petri Paavola*	**W02** Troubleshooting Sysinternals Tools 2018 Edition - *Sami Laiho*	**W03** In-Depth Introduction to Docker - *Neil Peterson*	**W04** How Microsoft Cloud Can Support Your GDPR Journey - *Milad Aslaner*
9:30 AM	10:45 AM	**W05** Automated Troubleshooting Techniques in Enterprise Domains (Part 2) - *Petri Paavola*	**W06** What's New in Windows Server 1803 - *Dave Kawula*	**W07** Simplify and Streamline Office 365 Deployments the Easy Way - *John O'Neill, Sr.*	**W08** How to Administer Microsoft Teams Like a Boss - *Ståle Hansen*
11:00 AM	12:00 PM	**TECHMENTOR PANEL: The Future of Windows** - *Peter De Tender, Dave Kawula, Sami Laiho, & Petri Paavola*			
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch - *McKinley* / Visit Exhibitors - *Foyer*			
1:00 PM	1:30 PM	Networking Break - Exhibitor Raffle @ 1:10 pm (Must be present to win) - *Foyer in front of Business Center*			
1:30 PM	2:45 PM	**W09** Putting the Windows Assessment and Deployment Kit to Work - *John O'Neill, Sr.*	**W10** Deploying Application Whitelisting on Windows Pro or Enterprise - *Sami Laiho*	**W11** Azure is 100% High-Available... Or Is It? - *Peter De Tender*	**W12** What the NinjaCat Learned from Fighting Cybercrime - *Milad Aslaner*
3:00 PM	4:15 PM	**W13** The Evolution of a Geek—Becoming an IT Architect - *Mike Nelson*	**W14** Advanced DNS, DHCP and IPAM Administration on Windows Server 2016 - *Orin Thomas*	**W15** Managing Tesla Vehicles from the Cloud - *Marcel Zehner*	**W16** Nano Server—Containers in the Cloud - *David O'Brien*
6:15 PM	9:00 PM	Set Sail! TechMentor's Seattle Sunset Cruise - *Buses depart the Hyatt Regency at 6:15pm to travel to Kirkland City Dock*			
START TIME	**END TIME**	**TechMentor Day 3: Thursday, August 9, 2018**			
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast			
8:00 AM	9:15 AM	**TH01** Manage Your Apple Investments with Microsoft EMS - *Peter Daalmans*	**TH02** Tips and Tricks for Managing and Running Ubuntu/Bash/Windows Subsystem for Linux - *Orin Thomas*	**TH03** The OMS Solutions Bakery - *Marcel Zehner*	**TH04** Getting Started with PowerShell for Office 365 - *Vlad Catrinescu*
9:30 AM	10:45 AM	**TH05** HoloLens, Augmented Reality, and IT - *John O'Neill, Sr.*	**TH06** A Real-world Social Engineering Attack and Response - *Milad Aslaner*	**TH07** 30 Terrible Habits of Server and Cloud Administrators - *Orin Thomas*	**TH08** Advanced PowerShell for Office 365 - *Vlad Catrinescu*
11:00 AM	12:15 PM	**TH09** 10 Tips to Control Access to Corporate Resources with Enterprise Mobility + Security - *Peter Daalmans*	**TH10** What's New and Trending with Microsoft Enterprise Client Management - *Kent Agerlund*	**TH11** OneNote LifeHack: 5 Steps for Succeeding with Personal Productivity - *Ståle Hansen*	**TH12** Managing Virtual Machines on AWS—Like in Real Life! - *David O'Brien*
12:15 PM	2:15 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center			
2:15 PM	3:30 PM	**TH13** Security Implications of Virtualizing Active Directory Domain Controllers - *Sander Berkouwer*	**TH14** Building a New Career in 5 Hours a Week - *Michael Bender*	**TH15** Azure CLI 2.0 Deep Dive - *Neil Peterson*	**TH16** OpenSSH for Windows Pros - *Anthony Nocentino*
3:45 PM	5:00 PM	**TH17** Running Hyper-V in Production for 10 years - Notes from the Field - *Dave Kawula*	**TH18** Network Sustainability and Cyber Security Measures - *Omar Valerio*	**TH19** Azure AD Connect Inside and Out - *Sander Berkouwer*	**TH20** I Needed to Install 80 SQL Servers...Fast. Here's How I Did It! - *Anthony Nocentino*
START TIME	**END TIME**	**TechMentor Post-Conference Workshops: Friday, August 10, 2018** *(Separate entry fee required)*			
8:30 AM	9:00 AM	Post-Conference Workshop Registration - Coffee and Light Breakfast			
9:00 AM	12:00 PM	**F01** Workshop: Hardening Your Windows Server Environment - *Orin Thomas*	**F02** Workshop: Learn the Latest and Greatest Updates to the Azure Platform IaaS and PaaS Services v2.0 - *Peter De Tender*		
12:00 PM	1:00 PM	Lunch - *McKinley*			
1:00 PM	4:00 PM	**F01** Workshop: Hardening Your Windows Server Environment (Continues) - *Orin Thomas*	**F02** Workshop: Learn the Latest and Greatest Updates to the Azure Platform IaaS and PaaS Services v2.0 (Continues) - *Peter De Tender*		

Speakers and sessions subject to change

CONNECT WITH TECHMENTOR

Twitter @TechMentorEvent

Facebook Search "TechMentor"

LinkedIn Search "TechMentor"

techmentorevents.com/redmond

Neural Regression Using CNTK

The goal of a regression problem is to make a prediction where the value to predict is a single numeric value. For example, you might want to predict the height of a person based on their weight, age and sex. There are many techniques that can be used to tackle a regression problem. In this article I'll explain how to use the CNTK library to create a neural network regression model.

A good way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo program creates a regression model for the well-known Yacht Hydrodynamics Data Set benchmark. The goal is to predict a measure of resistance for a yacht hull, based on six predictor variables: center of buoyancy of the hull, prismatic coefficient, length-displacement ratio, beam-draught ratio, length-beam ratio and Froude number.

The demo program creates a neural network with two hidden layers, each of which has five processing nodes. After training, the model is used to make predictions for two of the data items. The first item has predictor values (0.52, 0.79, 0.55, 0.41, 0.65, 0.00). The predicted hull resistance is 0.0078 and the actual resistance is 0.0030. The second item has predictor values (1.00, 1.00, 0.55, 0.56, 0.46, 1.00). The predicted hull resistance is 0.8125 and the actual resistance is 0.8250. The model appears to be quite accurate.

This article assumes you have intermediate or better programming skills but doesn't assume you know much about CNTK or neural networks. The demo is coded using Python, the default language for machine learning, but even if you don't know Python you should be able to follow along without too much difficulty. The code for the demo program is presented in its entirety in this article. The yacht hull data file used by the demo program can be found at bit.ly/2Ibsm5D, and is also available in the download that accompanies this article.

Understanding the Data

When creating a machine learning model, data preparation is almost always the most time-consuming part of the project. The raw data set has 308 items and looks like:

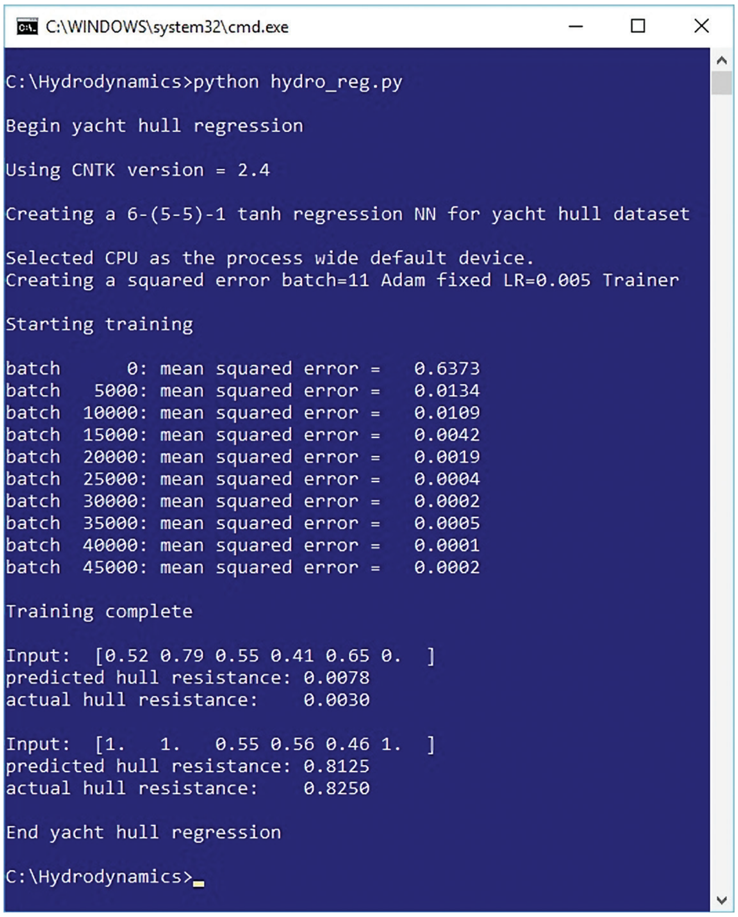
```
-2.3 0.568 4.78 3.99 3.17 0.125 0.11
-2.3 0.568 4.78 3.99 3.17 0.150 0.27
...
-5.0 0.530 4.78 3.75 3.15 0.125 0.09
...
-2.3 0.600 4.34 4.23 2.73 0.450 46.66
```

Code download available at msdn.com/magazine/0518magcode.

The file is space-delimited. The first six values are the predictor values (often called features in machine learning terminology). The last value on each line is the "residuary resistance per unit weight of displacement."

Because there's more than one predictor variable, it's not possible to show the complete data set in a graph. But you can get a rough idea of the structure of the data by examining the graph in **Figure 2**. The graph plots just the prismatic coefficient predictor values and the hull resistance. You can see that the prismatic coefficient values, by themselves, don't give you enough information to make an accurate prediction of hull resistance.

When working with neural networks, it's usually necessary to normalize the data in order to create a good prediction model. I used



```
C:\WINDOWS\system32\cmd.exe

C:\Hydrodynamics>python hydro_reg.py

Begin yacht hull regression

Using CNTK version = 2.4

Creating a 6-(5-5)-1 tanh regression NN for yacht hull dataset

Selected CPU as the process wide default device.
Creating a squared error batch=11 Adam fixed LR=0.005 Trainer

Starting training

batch      0: mean squared error =   0.6373
batch   5000: mean squared error =   0.0134
batch  10000: mean squared error =   0.0109
batch  15000: mean squared error =   0.0042
batch  20000: mean squared error =   0.0019
batch  25000: mean squared error =   0.0004
batch  30000: mean squared error =   0.0002
batch  35000: mean squared error =   0.0005
batch  40000: mean squared error =   0.0001
batch  45000: mean squared error =   0.0002

Training complete

Input:  [0.52 0.79 0.55 0.41 0.65 0.  ]
predicted hull resistance: 0.0078
actual hull resistance:    0.0030

Input:  [1.   1.   0.55 0.56 0.46 1.  ]
predicted hull resistance: 0.8125
actual hull resistance:    0.8250

End yacht hull regression

C:\Hydrodynamics>_
```

Figure 1 **Regression Using a CNTK Neural Network**

Manipulating Documents?

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

Try GroupDocs APIs for FREE

GROUPDOCS
Document Manipulation APIs

Microsoft® .NET

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

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

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

GroupDocs.Comparison
Compare two documents and get a difference summary report.

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

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

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

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

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

min-max normalization on the six predictor values and on the hull resistance values. I dropped the raw data into an Excel spreadsheet and, for each column, I computed the max and min values. Then, for each column, I replaced every value v with (v - min) / (max - min). For example, the minimum prismatic coefficient value is 0.53 and the maximum value is 0.60. The first value in the column is 0.568 and it's normalized to (0.568 - 0.53) / (0.60 - 0.53) = 0.038 / 0.07 = 0.5429.

After installing Anaconda, you install CNTK as a Python package, not a standalone system, using the pip utility.

After normalizing, I inserted tags |predictors and |resistance into the Excel spreadsheet so the data can be easily read by a CNTK data reader object. Then I exported the data as a tab-delimited file. The resulting data looks like:

```
|predictors  0.540000  0.542857 . . |resistance  0.001602
|predictors  0.540000  0.542857 . . |resistance  0.004166
. . .
```

Alternatives to min-max normalization include z-score normalization and order-magnitude normalization.

The Demo Program

The complete demo program, with a few minor edits to save space, is presented in **Figure 3**. All normal error checking has been removed. I indent with two space characters instead of the usual four as a matter of personal preference and to save space. Note that the '\' character is used by Python for line continuation.

Installing CNTK can be a bit tricky. First you install the Anaconda distribution of Python, which contains the necessary Python interpreter, required packages such as NumPy and SciPy, plus useful utilities such as pip. I used Anaconda3 4.1.1 64-bit, which has Python 3.5. After installing Anaconda, you install CNTK as a Python package, not a standalone system, using the pip utility. From an ordinary shell, the command I used was:

```
>pip install https://cntk.ai/PythonWheel/CPU-Only/cntk-
2.4-cp35-cp35m-win_amd64.whl
```

The hydro_reg.py demo has one helper function, create_reader. You can consider create_reader as boilerplate for a CNTK regression problem. The only thing you'll need to change in most scenarios is the tag names in the data file.

All control logic is in a single main function. The code begins:

```
def main():
  print("Begin yacht hull regression \n")
  print("Using CNTK version = " + \
    str(C.__version__) + "\n")
  input_dim = 6  # center of buoyancy, etc.
  hidden_dim = 5
  output_dim = 1  # residuary resistance
  train_file = ".\\Data\\hydro_data_cntk.txt"
...
```

Because CNTK is young and under continuous development, it's a good idea to display the version that's being used (2.4 in this case). The number of input nodes is determined by the structure of the data set. For a regression problem, the number of output nodes is always set to 1. The number of hidden layers and the number of processing nodes in each hidden layer are free parameters—they must be determined by trial and error.

The demo program uses all 308 items for training. An alternative approach is to split the data set into a training set (typically 80 percent of the data) and a test set (the remaining 20 percent). After training, you can compute loss and accuracy metrics of the model on the test data set to verify that the metrics are similar to those on the training data.

Creating the Neural Network Model

The demo sets up CNTK objects to hold the predictor and true hull resistance values:

```
X = C.ops.input_variable(input_dim, np.float32)
Y = C.ops.input_variable(output_dim)
```

CNTK uses 32-bit values by default because 64-bit precision is rarely needed. The name of the input_variable function can be a bit confusing if you're new to CNTK. Here, the "input_" refers to the fact that the return objects hold values that come from the input data (that correspond to both input and output of the neural network).

The neural network is created with these statements:

```
print("Creating a 6-(5-5)-1 NN")
with C.layers.default_options():
  hLayer1 = C.layers.Dense(hidden_dim,
    activation=C.ops.tanh, name='hidLayer1')(X)
  hLayer2 = C.layers.Dense(hidden_dim,
    activation=C.ops.tanh, name='hidLayer2')(hLayer1)
  oLayer = C.layers.Dense(output_dim,
    activation=None, name='outLayer')(hLayer2)
model = C.ops.alias(oLayer)  # alias
```

There's quite a bit going on here. The Python "with" statement can be used to pass a set of common parameter values to multiple functions. In this case, the neural network weights and biases values are initialized using CNTK default values. Neural networks are highly sensitive to initial weights and biases values, so supplying non-default values is one of the first things to try when your neural network fails to learn—a painfully common situation.

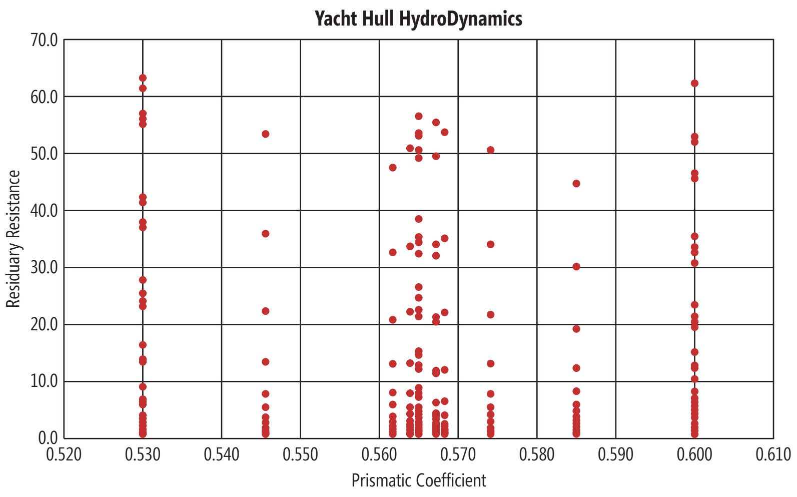


Figure 2 **Partial Yacht Hull Data**

Spreadsheets Everywhere.

SpreadsheetGear 2017 Released

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

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

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

Scalable Reporting

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

WIN Windows Forms | Silverlight | WPF WPF

Powerful Controls

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

Comprehensive Charting

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

Fastest Calculations

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

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

The neural network has two hidden layers. The X object as acts the input to the first hidden layer; the first hidden layer acts as input to the second hidden layer; and the second hidden layer acts as input to the output layer.

The heart of CNTK functionality is the ability to train a neural network model.

The two hidden layers use tanh (hyperbolic tangent) activation. The two main alternatives are logistic sigmoid and rectified linear units (ReLU) activation. The output layer uses the "None" activation, which means the values of the output nodes aren't modified. This is the design pattern to use for a regression problem. Using no activation is sometimes called using the identify activation function because the mathematical identity function is f(x) = x, which has no effect.

The demo program creates an alias named "model" for the output layer. This technique is optional and is a bit subtle. The idea here is that a neural network is essentially a complex math function. The output nodes conceptually represent both a layer of the network and the network/model as a whole.

Training the Model

The heart of CNTK functionality is the ability to train a neural network model. Training is prepared with these statements:

```
tr_loss = C.squared_error(model, Y)
max_iter = 50000
batch_size = 11
learn_rate = 0.005
learner = C.adam(model.parameters, learn_rate, 0.99)
trainer = C.Trainer(model, (tr_loss), [learner])
```

Figure 3 **Regression Demo Program**

```
# hydro_reg.py
# CNTK 2.4 with Anaconda 4.1.1 (Python 3.5, NumPy 1.11.1)
# Predict yacht hull resistance based on six predictors

import numpy as np
import cntk as C

def create_reader(path, input_dim, output_dim, rnd_order,
  sweeps):
  x_strm = C.io.StreamDef(field='predictors',
    shape=input_dim, is_sparse=False)
  y_strm = C.io.StreamDef(field='resistance',
    shape=output_dim, is_sparse=False)
  streams = C.io.StreamDefs(x_src=x_strm, y_src=y_strm)
  deserial = C.io.CTFDeserializer(path, streams)
  mb_src = C.io.MinibatchSource(deserial,
    randomize=rnd_order, max_sweeps=sweeps)
  return mb_src

# ========================================================

def main():
  print("\nBegin yacht hull regression \n")
  print("Using CNTK version = " + \
    str(C.__version__) + "\n")

  input_dim = 6  # center of buoyancy, etc.
  hidden_dim = 5
  output_dim = 1  # residuary resistance

  train_file = ".\\Data\\hydro_data_cntk.txt"

  # data resembles:
  # |predictors 0.540  0.542 . . |resistance  0.001
  # |predictors 0.540  0.542 . . |resistance  0.004

  # 1. create neural network model
  X = C.ops.input_variable(input_dim, np.float32)
  Y = C.ops.input_variable(output_dim)

  print("Creating a 6-(5-5)-1 tanh regression NN for \
yacht hull dataset ")
  with C.layers.default_options():
    hLayer1 = C.layers.Dense(hidden_dim,
      activation=C.ops.tanh, name='hidLayer1')(X)
    hLayer2 = C.layers.Dense(hidden_dim,
      activation=C.ops.tanh, name='hidLayer2')(hLayer1)
    oLayer = C.layers.Dense(output_dim,
      activation=None, name='outLayer')(hLayer2)
  model = C.ops.alias(oLayer)  # alias

  # 2. create learner and trainer
  print("Creating a squared error batch=11 Adam \
fixed LR=0.005 Trainer \n")
  tr_loss = C.squared_error(model, Y)

  max_iter = 50000
  batch_size = 11
  learn_rate = 0.005

  learner = C.adam(model.parameters, learn_rate, 0.99)
  trainer = C.Trainer(model, (tr_loss), [learner])

  # 3. create reader for train data
  rdr = create_reader(train_file, input_dim, output_dim,
    rnd_order=True, sweeps=C.io.INFINITELY_REPEAT)
  hydro_input_map = {
    X : rdr.streams.x_src,
    Y : rdr.streams.y_src
  }

  # 4. train
  print("Starting training \n")
  for i in range(0, max_iter):
    curr_batch = rdr.next_minibatch(batch_size,
      input_map=hydro_input_map)
    trainer.train_minibatch(curr_batch)
    if i % int(max_iter/10) == 0:
      mcee = trainer.previous_minibatch_loss_average
      print("batch %6d: mean squared error = %8.4f" % \
        (i, mcee))
  print("\nTraining complete")

  # (could save model to disk here)

  # 5. use trained model to make some predictions
  np.set_printoptions(precision=2, suppress=True)

  inpts = np.array(
    [[0.520000, 0.785714, 0.550000, 0.405512, \
      0.648352, 0.000000],
     [1.000000, 1.000000, 0.550000, 0.562992, \
      0.461538, 1.000000]],
    dtype=np.float32)

  actuals = np.array([0.003044, 0.825028],
    dtype=np.float32)

  for i in range(len(inpts)):
    print("\nInput: ", inpts[i])
    pred = model.eval(inpts[i])
    print("predicted resistance: %0.4f" % pred[0][0])
    print("actual resistance:    %0.4f" % actuals[i])

  print("\nEnd yacht hull regression ")

# ========================================================

if __name__ == "__main__":
  main()
```

A loss (error) function is required so the training object knows how to adjust weights and biases to reduce error. CNTK 2.4 has nine loss functions, but the simple squared_error is almost always suitable for a regression problem. The number of iterations corresponds to the number of update operations and must be determined by trial and error.

The Trainer object requires a Learner object. You can think of a Learner as an algorithm. CNTK supports eight learning algorithms. For regression problems, I typically get good results using either basic stochastic gradient descent or the more sophisticated Adam ("adaptive momentum estimation").

The batch size is used by CNTK to determine how often to perform weight and bias updates. The demo sets the batch size to 11. Therefore, the 308 items will be grouped into 308 / 11 = 28 randomly selected batches. Each batch is analyzed and then updates are performed. The learning rate controls the magnitude of the weight and bias adjustments. Determining good values for the batch size, the maximum number of iterations, and the learning rate are often the biggest challenges when creating a neural network prediction model.

The demo calls the program-defined create_reader function to, well, create a reader object. And an input_map is created that tells the reader where the feature values are and where the value-to-predict is:

```
rdr = create_reader(train_file, input_dim, output_dim,
  rnd_order=True, sweeps=C.io.INFINITELY_REPEAT)
hydro_input_map = {
  X : rdr.streams.x_src,
  Y : rdr.streams.y_src
}
```

The rnd_order parameter ensures that the data items will be processed differently on each pass, which is important to prevent training from stalling out. The INFINITELY_REPEAT argument allows training over multiple passes through the 308-item data set.

After preparation, the model is trained like so:

```
for i in range(0, max_iter):
  curr_batch = rdr.next_minibatch(batch_size,
    input_map=hydro_input_map)
  trainer.train_minibatch(curr_batch)
  if i % int(max_iter/10) == 0:
    mcee = trainer.previous_minibatch_loss_average
    print("batch %6d: mean squared error = %8.4f" % \
      (i, mcee))
```

The next_minibatch function pulls 11 items from the data. The train function uses the Adam algorithm to update weights and biases based on squared error between computed hull resistance values and actual resistance values. The squared error on the current 11-item batch is displayed every 50,000 / 10 = 5,000 batches so you can visually monitor training progress: You want to see loss/error values that generally decrease.

Using the Model

After the model has been trained, the demo program makes some predictions. First, the predictor values for two arbitrary items from the normalized data set are selected (items 99 and 238) and placed into an array-of-arrays style matrix:

```
inpts = np.array(
  [[0.520000, 0.785714, 0.550000, 0.405512,
    0.648352, 0.000000],
   [1.000000, 1.000000, 0.550000, 0.562992,
    0.461538, 1.000000]],
  dtype=np.float32)
```

Next, the corresponding normalized actual hull resistance values are placed into an array:

```
actuals = np.array([0.003044, 0.825028], dtype=np.float32)
```

Then, the predictor values are used to compute the predicted values using the model.eval function, and predicted and actual values are displayed:

```
for i in range(len(inpts)):
  print("\nInput: ", inpts[i])
  pred = model.eval(inpts[i])
  print("predicted resistance: %0.4f" % pred[0][0])
  print("actual resistance:    %0.4f" % actuals[i])

print("End yacht hull regression ")
```

When creating a neural network regression model, there's no predefined accuracy metric.

Notice that the predicted hull resistance value is returned as an array-of-arrays matrix with a single value. Therefore, the value itself is at [0][0] (row 0, column 0). Dealing with shapes of CNTK vectors and matrices is a significant syntax challenge. When working with CNTK I spend a lot of time printing objects and displaying their shape, along the lines of print(something.shape).

Wrapping Up

When creating a neural network regression model, there's no predefined accuracy metric. If you want to compute prediction accuracy you must define what it means for a predicted value to be close enough to the corresponding actual value in order to be considered correct. Typically, you'd specify a percentage/proportion, such as 0.10, and evaluate a predicted value as correct if it's within that percentage of the actual value.

Because the demo model works with normalized data, if you use the model to make a prediction for new, previously unseen predictor values, you have to normalize them using the same min-max values that were used on the training data. Similarly, a predicted hull resistance value, pv, is normalized, so you'd have to de-normalize by computing pv * (max - min) + min.

The term "regression" can have several different meanings. In this article the term refers to a problem scenario where the goal is to predict a single numeric value (hull resistance). The classical statistics linear regression technique is much simpler than neural network regression, but usually much less accurate. The machine learning logistic regression technique predicts a single numeric value between 0.0 and 1.0, which is interpreted as a probability and then used to predict a categorical value such as "male" ($p < 0.5$) or "female" ($p > 0.5$). ■

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

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

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

JOIN US AT

The Ultimate Education Destination

2018 Orlando

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

DECEMBER 2-7, 2018

Live! 360: A Unique Conference for the IT and Developer Community

- 6 FULL Days of Training Including Hands-On Labs & Workshops
- 6 Co-Located Conferences for 1 Low Price
- Customize Your Own Agenda from Hundreds of Sessions
- Expert Education and Training
- Knowledge Share and Networking

CONNECT WITH LIVE! 360

twitter.com/live360
@live360

facebook.com
Search "Live 360"

linkedin.com
Join the "Live! 360" group!

instagram.com
@live360_events

EVENT PARTNERS
Microsoft
Magenic

SILVER SPONSOR
Progress

SUPPORTED BY
msdn magazine
Redmond Channel Partner
Redmond MAGAZINE

HANDS-ON LABS

Join us for a full-day of pre-conference Hand-On Labs on Sunday, December 2.

SUMMER SAVINGS = BEST SAVINGS!

REGISTER NOW

REGISTER BY 8/31 AND SAVE $500!

Use promo code: MSDN

See website for details.

6 CO-LOCATED CONFERENCES, 1 GREAT PRICE!

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio Live! features unbiased and practical development training on the Microsoft Platform. Come join us and code in paradise!

SQL Server LIVE! TRAINING FOR DBAs AND IT PROS

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

TECHMENTOR IN-DEPTH TRAINING FOR IT PROS

TechMentor is where IT training meets sunshine, with zero marketing speak on topics you need training on now, and solid coverage on what's around the corner.

NEW! IN 2018

Artificial Intelligence LIVE! AI FOR DEVELOPERS AND DATA SCIENTISTS

Artificial Intelligence Live! is an innovative, new conference for current and aspiring developers, data scientists, and data engineers covering artificial intelligence (AI), machine learning, data science, Big Data analytics, IoT & streaming analytics, bots, and more.

Office & SharePoint LIVE! ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live! provides leading-edge knowledge and training to work through your most pressing projects and enable people to work from anywhere at any time.

Modern Apps LIVE! MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Modern Apps Live!, presented in partnership with Magenic, focuses on how to architect, design and build a complete Modern App from start to finish.

VIRTUALIZATION & Cloud Review

Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA INC YOUR GROWTH. OUR BUSINESS.

LIVE360EVENTS.COM

Ol' Man River

I just paid my income taxes, so I'm feeling cranky. To cheer myself up, I'm going to kick over my all-time favorite hornets' nest: Visual Basic 6. My three previous columns on it (msdn.com/magazine/jj133828, msdn.com/magazine/dn745870 and msdn.com/magazine/mt632280) have generated far more mail, pro and con, than anything else I've ever written. Once again, I'll goad the developers who continue to love VB6, and those who love to hate it and them, into spectacular combat, for my amusement and yours. Damn, this is fun.

VB6 just got an important boost from Microsoft blogger Scott Hanselman. In his post (bit.ly/2rcPD0f), Hanselman shows how to configure a VB6 app to be hosted in the Windows 10 Store, using the Microsoft Desktop Bridge infrastructure and tools (bit.ly/2HFVzcc). That's huge, as hosting an app in the store means that Microsoft is at least somewhat vouching for its compatibility and content. Potential purchasers perceive it as sort of a Good Computing Seal™—perhaps not as strong as Apple's, but definitely much stronger than Google's. You may have to modify your app somewhat to meet the store's policies (bit.ly/2HHUXiq), such as removing "excessive or gratuitous profanity." (Well, %*&#$ that, I say. Oops.) But this should be relatively easy.

I can imagine VB6 apps doing things like updating live tiles. These life-extenders should drive VB6 detractors barking mad.

These bridging tools instruct Windows 10 to enforce good behavior on regular, not-otherwise-compliant, Win32 apps. For example, Windows 10 (when properly instructed) will use a separate registry file to handle changes the app might make to the system registry, so it can't clobber any other apps or resources. For another example, any changes the app might make to the file system are automatically redirected to the ApplicationData.LocalFolder, where Windows 10 standards require them to reside. You can see this strategy at bit.ly/2I3n0fG.

But wait! There's more! Microsoft has exposed many of the Windows 10 salient features to native Win32 apps (see bit.ly/2JFPgSI). VB6 apps are, by definition, native Win32 apps. It's only a matter of time until someone writes a COM bridge to Universal Windows Platform, so VB6 can use it easily. I can imagine VB6 apps doing things like updating live tiles. These life-extenders should drive VB6 detractors barking mad.

Maybe this is why Microsoft won't release VB6 as open source, as it has for most of its tools. It might be worried that the community would change it to the point that Microsoft couldn't provide this "It Just More-or-Less Works" (IJM-o-LW) compatibility in the future.

I rarely use VB6 for commercial software development, as its tradeoffs are not usually the right set for my clients today. But I do have it installed on my experimental network for testing. I have a big problem (not an issue, see my old column on "Weasel Words," msdn.com/magazine/ff955613) with people who have a big problem with other developers' choices. Why do you care what someone else uses? Are you a Puritan as H.L. Mencken describes them: someone who lies awake at night with the haunting fear that someone, somewhere, may be happy?

VB6 programmers chose a different set of tradeoffs than you did. Yes, you get frustrated, virtuously slogging through infrastructure, while they ignore scalability and robustness and plunge merrily ahead. No, they probably don't understand the underlying COM very well—almost nobody does these days. When they inevitably get in trouble, I'll bail them out (for a fee, of course, see graybeardsoftware.com). That's their call. Mind your own damn business.

With these latest improvements to compatibility, I foresee at least another ten years of life for VB6. And I'll bet you anything that this support gets renewed in Windows 11 and 12, or whatever they're called by then. Another decade of driving the puritans crazy. I can dig it.

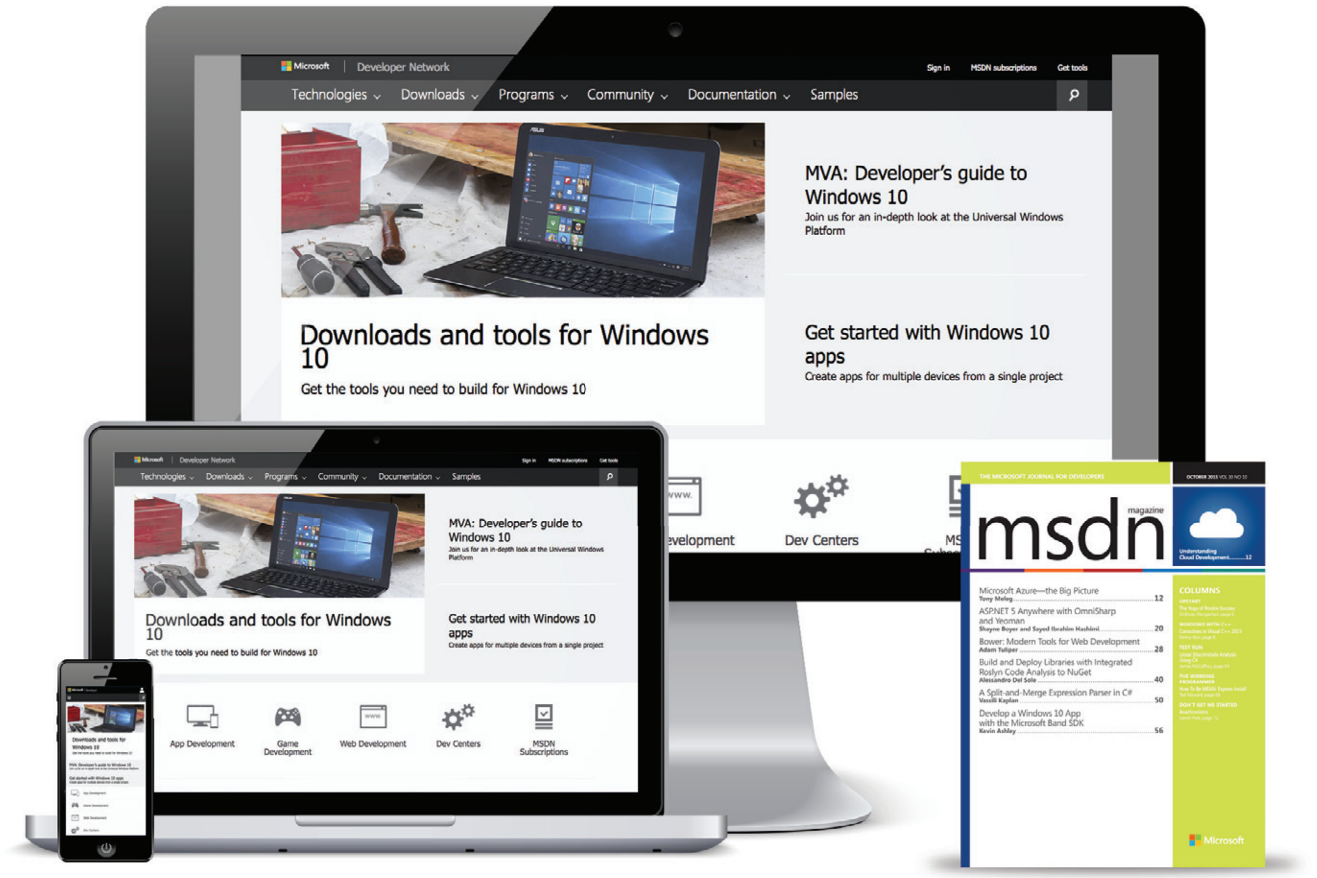
I've likened VB6 to a cockroach, a bus and a knuckleball. Today VB6 continues to cut a path to working apps, eroding its way through new obstacles, as the Mississippi River cuts new pathways through its delta to the sea, even as the silt it carries clogs the old ones. Like Ol' Man River, VB6 just keeps rollin' along.

Note: a beautiful clip of this song, sung by Paul Robeson in James Whale's classic 1936 film version of "Show Boat," is online at bit.ly/2JJFv66. It's worth a listen. ■

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

msdn
magazine

Where you need us most.



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

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

Learn, Explore, Use

Your Destination for Data Cleansing & Enrichment APIs

Global Address
Global Email
Global Phone
Global Name
melissa
DEVELOPER
ID Verification
Global IP Locator
Property Data
Business Coder

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

RAPID APPLICATION DEVELOPMENT

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

REAL-TIME & BATCH PROCESSING

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

TRY OR BUY

Easy payment options to free funds for core business operations.

FLEXIBLE CLOUD APIS

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

Turn Data into Success – Start Developing Today!

Melissa.com/developer

1-800-MELISSA

melissa