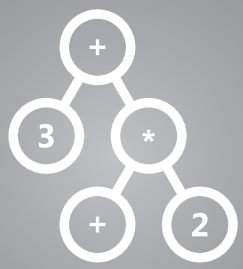
THE MICROSOFT JOURNAL FOR DEVELOPERS

SEPTEMBER 2019 VOL 34 NO 9

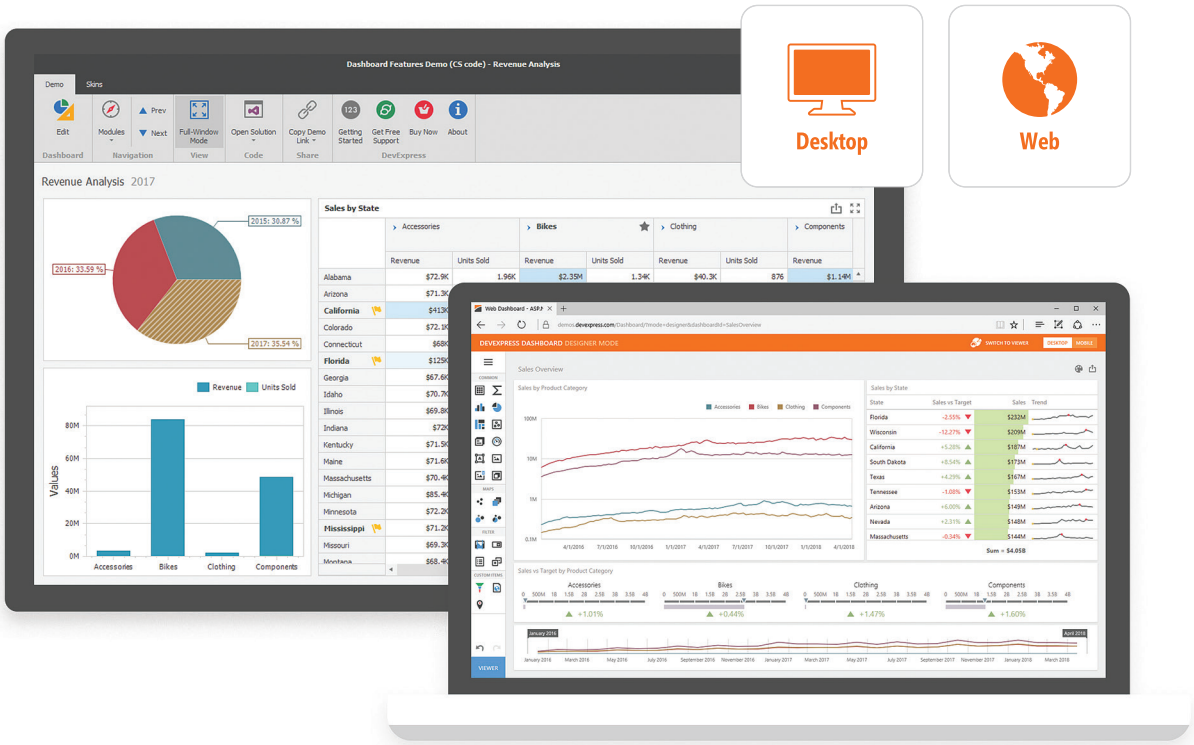
msdn magazine



DevExpress®

DevExpress Dashboard for .NET

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



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

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

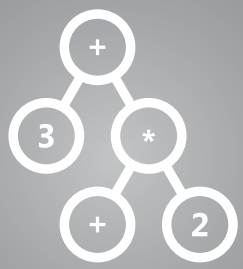
Learn More Today — Try DevExpress Dashboard Risk Free for 30 days

devexpress.com/dashboard

THE MICROSOFT JOURNAL FOR DEVELOPERS

SEPTEMBER 2019 VOL 34 NO 9

msdn magazine



COLUMNS

Microsoft

INFRAGISTICS

Faster Paths to Amazing Experiences

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

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

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

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

Delivering amazing experiences for 30 years.

Infragistics Ultimate

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

msdn magazine

SSEPTEMBER 2019 VOLUME 34 NUMBER 9

ID STATEMENT *MSDN Magazine* (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in December by 1105 Media, Inc., 6300 Canoga Avenue, Suite 1150, Woodland Hills, CA 91367. Periodicals postage paid at Woodland Hills, CA 91367 and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: *MSDN Magazine*, File 2272, 1801 W.Olympic Blvd., Pasadena, CA 91199-2272, email MSDNmag@1105service.com or call 866-293-3194 or 847-513-6011 for U.S. & Canada; 00-1-847-513-6011 for International, fax 847-763-9564. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

CORPORATE ADDRESS 1105 Media, Inc.
6300 Canoga Avenue, Suite 1150, Woodland Hills 91367
1105media.com

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

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

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

Reaching the Staff

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

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

Irvine Office (weekdays, 9:00 a.m.-5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
2121 Alton Pkwy., Suite 240, Irvine, CA 92606

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

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

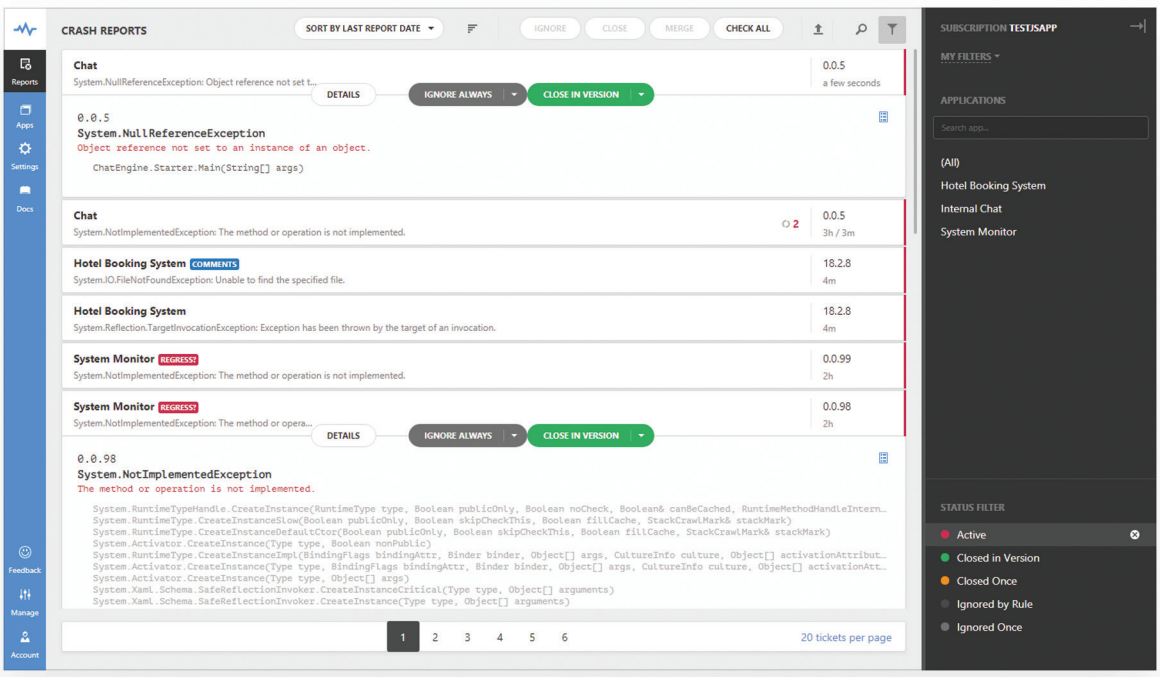
Microsoft

BPA WORLDWIDE
1906 2006 AMERICAN BUSINESS MEDIA
WPA Western Publications Association

logify™

Detect Errors and Exceptions
with 24/7 Application Monitoring

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



Learn More Today — Try Logify Risk Free for 15 days

devexpress.com/logify

DevExpress®

MICHAEL DESMOND

All Good Things

If you haven't heard the news already, I'll just come out and say it: Microsoft has announced that *MSDN Magazine* will close its doors after the publication of its November issue.

As you can imagine, this is an emotional moment, both for the editors, columnists and authors who work on each issue of the magazine, and for the thousands of developers who rely on it to stay up-to-date with Microsoft's evolving tool stack. For 33 years, going back to its origins as *Microsoft Systems Journal* (MSJ) in 1986, this publication has provided hands-on technical guidance and insight to developers. Charles Petzold taught us how to program for Windows, Dino Esposito untangled the complexities of ASP.NET and James McCaffrey illuminated the workings of Microsoft's fast-changing machine learning tool stack.

From MS-DOS to Microsoft Azure, the scope and scale of our work has certainly changed, but the role of *MSDN Magazine* in enabling that work never wavered. The magazine has been there to shine a light in the darkness—to make clear the opaque nature of new tooling, to render simplicity out of complexity, and to provide inspiration to developers anxious to learn, master and create.

But it's emerged also as living history. Scan the covers of old issues of *MSDN Magazine* and MSJ before it, and you see the parade of products, tools, frameworks and languages that have populated and defined the Microsoft ecosystem over the years. In providing timely code-level guidance to developers, the magazine chronicled the evolution of Microsoft from its days as a PC-focused upstart.

MSDN Magazine will soon be history, but that doesn't mean the important work of enabling and educating developers will stop. Far from it, Microsoft has worked to unify and streamline its code-level guidance around the docs.microsoft.com portal, which provides a comprehensive repository of wisdom for developers across the Microsoft ecosystem. Microsoft also intends to migrate archived *MSDN Magazine* content to docs.microsoft.com, so that wisdom, too, will be discoverable on the site.

From MS-DOS to Microsoft Azure, the scope and scale of our work has certainly changed, but the role of *MSDN Magazine* in enabling that work never wavered.

They say all good things must come to an end. And so, the long story of *MSDN Magazine* draws at last to a close, and the common thread it wove among developers across three decades of breakneck advancement closes with it. I would say it's been my honor and privilege to helm *MSDN Magazine* for the past eight years, but I know that the work remains unfinished. We have issues to publish in October and November, and they will carry forward the full spirit, tradition and mandate of *MSDN Magazine* going back to 1986—to provide actionable, code-level guidance to developers using Microsoft tools, frameworks and languages.

I'll see all of you next month.

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

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

LEADTOOLS
THE WORLD LEADER IN IMAGING SDKs

Powerful DICOM, PACS, and HL7 functionality.

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

Intellifront BI | from $23,246.35

Christiansteven

Cost-Effective Data Analytics & Business Intelligence.

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

DevExpress DXperience 19.1 | from $1,439.99

DevExpress

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

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

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

Christiansteven

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

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

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

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R

Stay tuned

NEO
smart economy

NEO + VS Code = ?

KATHLEEN DOLLARD

Visual Basic on .NET Core

I started writing Visual Basic code more than two decades ago, and I understand why so many people still program in Visual Basic .NET. It has almost all of the features of C#, plus unique functionality that makes it easier for you to focus on what your software accomplishes. This comes from features in the language itself, as well as from language extensions and language stability. Visual Basic also includes unique productivity features like XML literals and in-place event hookup.

Now Visual Basic .NET 16.0 is bringing your favorite Visual Basic features to .NET Core 3.0. The essential parts of the Visual Basic .NET programming language have been on .NET Core since its early development, but developers can expect an enriched experience when Microsoft ships Visual Studio 16.3 and .NET Core 3.0 with Visual Basic 16.0 and C# 8.0 on board.

> In my role working on the transition to .NET Core, I was able to dive into the technology behind the language extensions: the special functions, application models and My subsystem.

In my role working on the transition to .NET Core, I was able to dive into the technology behind the language extensions: the special functions, application models and My subsystem. These features are contained in microsoft.visualbasic.dll, also called the Visual Basic Runtime, and many are now included in .NET Core 3.0, except those dependent on Windows Forms (WinForms).

Windows Forms

Visual Basic .NET has a special relationship with WinForms, which was modeled largely on earlier versions of Visual Basic. Among all the options .NET programmers have to build applications, WinForms remains the best at getting the job done fast. In addition to traditional roles, WinForms offers a rapid way to develop thin front ends for services on-premises or in the cloud, either for production or for functional prototypes.

While the WinForms library will be available, the WinForms designers will not be part of Visual Studio 16.3. This limits the experience, so the Visual Basic .NET team decided to focus on the non-WinForms portion of the language extensions for Visual Basic 16.0. This means you can use WinForms on .NET Core with Visual Basic, but you won't have the project property dialog to enable the Visual Basic Application Model. You'll need either a Sub Main or a startup form, and you'll find that the My features aren't yet available.

Parts of the Visual Basic Runtime depend on WinForms, even for unexpected types like My.Computer. We're splitting the runtime into the parts that are dependent on WinForms and those that aren't, with the WinForms-dependent part to appear in a future release of Visual Basic.

Beyond these limitations, Visual Basic .NET 16.0 brings much of the functions of the Visual Basic runtime to .NET Core. This includes key features, like Fix and Mid, that you expect. Telemetry from API Port helped the team prioritize the work here, and some features with very low usage weren't ported.

Openness and Stability

Visual Basic .NET 16.0 includes the financial and file functions that were ported by folks in the community. Of course, Visual Basic .NET has been open source since 2015. There are significant areas where you can contribute, and many of them aren't nearly as intimidating as the Roslyn compiler! You can also be part of the revival of Visual Basic .NET communities on Facebook and Gitter. Find more about the community and language design at the Visual Basic .NET language design site (github.com/dotnet/vblang).

For this latest version of Visual Basic, the runtime was ported directly. There were no changes and no effort to "clean up" features. Things should work the same in .NET Core as they did in .NET Framework. All this is part of the deep commitment to stability within the Visual Basic team. This stability is important for backward compatibility, of course, but the commitment extends to ensuring that code written at different points in the evolution of Visual Basic continues to be easy to read. New features are incorporated slowly in Visual Basic .NET, and only those that feel natural in Visual Basic are added.

You can develop applications targeting either .NET Core or .NET Framework (.NET 4.8 and below) with Visual Studio. While .NET Framework will remain supported for a long time, developing applications on .NET Core brings a raft of advantages, including side-by-side and self-contained deployment that eliminates issues that occur when another application's installation makes changes

File Format APIs

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

Try risk free - 30 day trial

ASPOSE

Download a Free Trial at

https://downloads.aspose.com

Aspose.Words

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

Aspose.Cells

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

Aspose.Pdf

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

Aspose.Slides

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

Aspose.Email

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

Aspose.BarCode

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

Aspose.Imaging

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

Aspose.Tasks

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

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

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

to production machines. For WinForms, there are new features like better high DPI support. And going forward, new capabilities in .NET, Visual Basic .NET and C# will only be available on .NET Core. In Visual Basic, you'll get the advantages of Visual Basic 16.0 just by targeting .NET Core 3.0 (netcoreapp2.2).

Cross-Platform Support

Visual Basic .NET on .NET Core is cross-platform, although WinForms, Windows Presentation Foundation and other Windows-specific features will work only on Windows. You can find the OSes that are supported at aka.ms/net-core-3-0-supported-os. If you run a Visual Basic .NET application on an OS like Linux, features that work cross-platform will just work. If you call Visual Basic Runtime functionality that doesn't work on that platform, you'll get a System.PlatformNotSupportedException with a message similar to "<method> not supported on this platform." This is the same behavior as the rest of .NET, so if you're working cross-platform, you'll want to test your application across the OSes where you expect to deploy, regardless of the language you use.

Some project types aren't supported on .NET Core 3.0. For example, WebForms won't be supported in any language. Visual Basic isn't supported by ASP.NET Core Razor, so you can't simply port MVC applications. And while Microsoft doesn't offer a Web development model that's 100 percent Visual Basic, you can use Visual Basic in ASP.NET WebAPI with JavaScript front ends, or create combined apps with views in C# Razor projects.

Changes to Visual Studio and .NET Core Installers

If you run `dotnet --info` at a command prompt, you'll see a list of installed .NET Core SDKs and runtimes. There may be a lot more than you anticipated!

Earlier Visual Studio and .NET Core installers haven't been removing older SDKs and runtimes when they update or uninstall. While you may need these to support SDK pinning via global.json or to target older runtimes, they might just be sitting unused on your machine.

Now, starting with Visual Studio 2019 16.3, Visual Studio will manage the versions of .NET Core SDKs and runtimes it installs. It will only keep one copy of the .NET Core SDK on each machine per channel (preview or release), and will install the latest runtime. You can target earlier runtimes by selecting them—along with their templates and targeting packs—in the Individual Components tab of the Visual Studio Installer.

When you download and install the .NET Core 3.0 SDK from dotnet.microsoft.com/download, earlier patches in the same feature band will now be removed. For example, 3.0.100 will be uninstalled when you install 3.0.102. Previews in that band will also be removed.

Each version of the SDK can target all earlier versions of the runtime, so you generally only need one. If you need additional SDKs or runtimes, you can download them from dotnet.microsoft.com/download.

You can manually remove .NET Core SDKs and runtimes, or you can clean them up using the recently released .NET Core Uninstall Tool on Windows and macOS (aka.ms/remove-sdk-runtime). Just be careful, SDKs aren't tracked by Visual Studio, so removing the wrong ones can cause issues. If you delete something that Visual Studio needs, run "Repair" in the Visual Studio installer.

API Portability Anaylzer

You can test the compatibility of your applications by running the API Portability Analyzer. The tool is available for download as a Visual Studio extension from the Visual Studio gallery or as a command-line tool. Find out more at aka.ms/api-portability. The API Portability Analyzer outputs a spreadsheet listing the percent of your application that will just work in the platforms you select, in this case .NET Core 3.0. Other tabs drill into the specific APIs used in the application, as well as those that aren't supported.

We Want To Hear from You!

The team wants to understand the issues that face Visual Basic .NET programmers moving to .NET Core and we invite your help on the next stage of that journey. If you run the Portability Analyzer and find you need things missing in the VisualBasic namespace or other Visual Basic-specific issues, let us know by opening an issue, or by commenting on an existing one, at the Visual Basic .NET language design site (github.com/dotnet/vblang).

> While .NET Framework will remain supported for a long time, developing applications on .NET Core brings a raft of advantages, including side-by-side and self-contained deployment that eliminates issues that occur when another application's installation makes changes to production machines.

The work we're doing with .NET Core sets up Visual Basic for the future. Combined with Microsoft's long-term commitment to .NET Framework 4.8, you have flexibility for both new and legacy applications in Visual Basic, one of the most productive programming languages ever created. ■

Kathleen Dollard *is a principal program manager on the .NET Core team at Microsoft. She is the Program Manager for Visual Basic, contributes to the Managed Languages and works on the .NET Core CLI and SDK.*

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

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

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

Where In-Depth IT Training Meets the Sunshine

TechMentor offers in-depth training for IT Pros, giving you the perfect balance of the tools you need today, while preparing you for tomorrow. Expect troubleshooting tips, performance optimization training, and best practices from peers and experts in the industry. Plus, there will be dedicated coverage of Windows PowerShell, core Windows Server functionality, Security, System Center and so much more. We'll see you in the sun!

TRACK TOPICS INCLUDE:

- Client and Endpoint Management
- PowerShell and DevOps
- Infrastructure
- Soft Skills for ITPros
- Security and Ethical Hacking
- Azure (Public/Hybrid)
- Office 365 for the IT Pro

SAVE $400 With Super Early Bird Savings. Register by September 27!
Use Promo Code MSDN

A Part of Live! 360: The Ultimate Education Destination

6 Great Events, 1 Low Price!

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

techmentorevents.com/orlando

EVENT PARTNER: Microsoft
PLATINUM SPONSOR: accusoft
SILVER SPONSOR: aws
SUPPORTED BY: msdn magazine, Redmond Channel Partner, Redmond Magazine, Virtualization & Cloud Review, Visual Studio Magazine

AGENDA-AT-A-GLANCE

#LIVE360

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

START TIME	END TIME	
TechMentor Full Day Hands-On Labs: Sunday, November 17, 2019		
7:15 AM	9:00 AM	Registration • Coffee and Morning Pastries
9:00 AM	6:00 PM	**TMS01** Hands-On Lab: You Got the Azure Workshops Before, Now Learn it Hands-on!! - *Peter De Tender* | **TMS02** Hands-On Lab: Rock JEA (Just Enough Administration) the Easy Way with Windows Admin Center - *John O'Neill Sr.*
2:00 PM	7:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center
TechMentor Pre-Conference Workshops: Monday, November 18, 2019		
7:00 AM	8:30 AM	Registration • Coffee and Morning Pastries
8:30 AM	5:30 PM	**TMM01** Workshop: Azure from IAAS to PAAS—A Demo-Filled Workshop - *Koenraad Haedens* | **TMM02** Workshop: PowerShell Remoting Deep Dive - *Jeffery Hicks*
6:30 PM	8:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:30pm
TechMentor Day 1: Tuesday, November 19, 2019		
7:00 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:00 AM	**TECHMENTOR KEYNOTE: To Be Announced** - *Chris Jackson, Principal Architect, Windows Customer Experience Engineering, Microsoft*
9:15 AM	10:30 AM	**TMT01** PowerShell Tools and Techniques Basics for IT Pros, Not Devs - *John O'Neill Sr.* | **TMT02** Protecting Your Corporate Resources Using Azure Identity - *Peter De Tender* | **TMT03** Zero Trust: Trust Identities, Not Your Network - *Ricky Pullan & Swetha Rai*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - *Pacifica 7*
11:00 AM	12:00 PM	**LIVE! 360 KEYNOTE: The Power of Real World DevOps** - *Jessica Deen, Azure Avenger, Microsoft & Abel Wang, Senior Cloud Developer Advocate, Microsoft*
12:00 PM	12:45 PM	Lunch • Visit the EXPO
12:45 PM	1:30 PM	Dessert Break • Visit the EXPO
1:30 PM	1:50 PM	**TMT04** Fast Focus: The 10 Handiest PowerShell Cmdlets for Managing Hyper-V - *John O'Neill Sr.* | **TMT05** Fast Focus: Rock Configuration Manager in 20 Minutes - *Émile Cabot*
2:00 PM	2:20 PM	**TMT06** Fast Focus: PowerShell Fast & Furious - *Jeffery Hicks* | **TMT07** Fast Focus: Be a Better Speaker—How to Create Presentations that Inspire and Make People Care - *Erwin Derksen*
2:20 PM	2:45 PM	Networking Break • Visit the EXPO - Pacifica 7
2:45 PM	4:00 PM	**TMT08** An Inclusive Look at Fighting Phishing - *Roger Grimes* | **TMT09** Troubleshooting the Modern Managed Client - *Ronnie Pedersen* | **TMT10** Azure Windows Virtual Desktop – Deploy and Configure in 50 Minutes - Koenraad Haedens
4:15 PM	5:30 PM	**TMT11** Fast-Track Your Way to IT Pro Rockstar Status - *John O'Neill Sr. & Cristal Kawula* | **TMT12** Azure Security at Your Service (Learn Azure Security Center and Azure Sentinel) - *Peter De Tender* | **TMT13** Secure Remote Management with Just Enough Administration - *Jeffery Hicks*
5:30 PM	7:30 PM	Exhibitor Reception - Pacifica 7
TechMentor Day 2: Wednesday, November 20, 2019		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:15 AM	**TMW01** 10 Ways to Hack You Via Email - *Roger Grimes* | **TMW02** How YOU Deploy Office ProPlus / 2019 in the Enterprise Successfully - *Erwin Derksen* | **TMW03** Windows Powershell and Workflow: A Powerful Pairing! - *Michael Wiley*
9:30 AM	10:45 AM	**TMW04** Deploying and Managing Office 365 ProPlus in a Modern World - *Ronnie Pedersen* | **TMW05** Managing Android and IOS in the Enterprise - *Peter Daalmans* | **TMW06** Become a PowerShell Debugging Ninja! - *Kirk Munro*
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: To Be Announced**
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO
2:00 PM	3:15 PM	**TMW07** Surviving a Ransomware Attack Using Azure Site Recovery (Demo Fest) - *Émile Cabot & Dave Kawula* | **TMW08** Everything You Need to Know to Manage Modern IT Apps Using Intune - *Peter Daalmans* | **TMW09** Creating and Sharing Awesome PowerShell Modules: A No Nonsense Guide - *Kirk Munro*
3:15 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - Pacifica 7
4:00 PM	5:15 PM	**TMW10** Driving Adoption of the Modern Workplace - *Émile Cabot* | **TMW11** 5 Crucial Mistakes Made in IT-Projects (and How to Avoid Them) - *Erwin Derksen* | **TMW12** Life Beyond Functions: Creating Powershell Script Cmdlets (aka Advanced Functions) - *Michael Wiley*
7:30 PM	9:00 PM	Live! 360 Dessert Luau - Wantilan Pavilion
TechMentor Day 3: Thursday, November 21, 2019		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	9:15 AM	**TMH01** Build Your Azure Infrastructure like a Pro - *Aleksandar Nikolic* | **TMH02** Imposter Syndrome: Overcoming Self-Doubt in Success - *Heather Downing* | **TMH03** Windows Autopilot Deep Dive - *Petri Paavola*
9:30 AM	10:45 AM	**TMH04** How to Get Your Cloud Services Secured with Intune and Azure AD - *Peter Daalmans* | **TMH05** Best Tips to Make the Life of SCCM Admins Easier! - *Panu Saukko* | **TMH06** What I Learned When I Moved my Operations to the Cloud - *Sami Laiho*
11:00 AM	12:00 PM	**TECHMENTOR PANEL DISCUSSION: The Future of IT** *Moderated by Sami Laiho and Dave Kawula; Émile Cabot, Petri Paavola, & Panu Saukko*
12:00 PM	1:00 PM	Lunch - *Oceana Ballroom*
1:00 PM	2:15 PM	**TMH07** Building Real World Labs in Azure - *Dave Kawula* | **TMH08** Implementing Proactive Security in the Cloud - *Sami Laiho* | **TMH09** How to Use PowerShell to Become a Windows Management SuperHero—2019 Edition - *Petri Paavola*
2:30 PM	3:45 PM	**TMH10** Becoming a Community Rockstar - *Cristal Kawula* | **TMH11** Managing Azure VMs with Windows Admin Center - *Aleksandar Nikolic* | **TMH12** Ready, Set, Go: Start Managing Your Windows 10 Devices with Intune - *Panu Saukko*
4:00 PM	5:00 PM	**NEXT? LIVE! 360 NETWORKING EVENT** - *Bradley Ball, Andrew Brust, Ben Curry, Peter Daalmans, Joseph D'Antoni, Marcel de Vries, Billy Hollis, Dave Kawula, Raj Krishnan, Thomas Sami Laiho, LaRock, Vishwas Lele, Rockford Lhotka, Karen Lopez, Matthew McDermott, Agnes Molnar, Brian Randell, Panu Saukko, Jen Stirrup, & Alex Thissen*
TechMentor Post-Conference Workshops: Friday, November 22, 2019		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries
8:00 AM	5:00 PM	**TMF01** Workshop: Migrating and Upgrading to Windows Server 2019 - *Dave Kawula* | **TMF02** Workshop: How to Build Intune Managed Windows 10 with the Best User Experience - *Petri Paavola*

Speakers and sessions subject to change

PRODUCED BY CONVERGE360 an 1105 MEDIA company

CONNECT WITH LIVE! 360
twitter.com/live360 @live360
facebook.com Search "Live 360"
instagram.com @live360_events
linkedin.com Join the "Live! 360" group!

TED NEWARD

An Introduction to Python

In the last few years, Python has "made the mainstream" in terms of popular use, rated by some surveys and polls as the new most popular language. While it's doubtful that Python will ever take over the world, as they say, it does seem to have a relatively strong grip on the data science and artificial intelligence/machine learning world, and for that reason alone, Python is an interesting language to study.

With the combination of its relatively simple syntax and programming model, and the extensible modules connecting it to the underlying OS, Python makes for a timely and useful switch to a new topic for this column. Over the next few issues, I'll explore Python's syntax, semantics, a number of the more interesting libraries in the Python ecosystem, and then … Who knows?

First Things First

Before I dive into this exploration, however, I must present a necessary disclaimer: As is the case for almost all of the columns I write, I'm not going to suggest for even a half-moment that Python should replace your C# (or Visual Basic) code currently in existence. Python is a useful tool to have on your belt, just as so many other things are, and if you find yourself in a situation where it seems appropriate, use it. But there's nothing I can think of that you can do with Python that can't also be done with C#—you just use them differently, that's all.

> In fact, the Python appellation derives from the other famous Python: Monty.

Additionally, I would like to clear up a popular misconception: Despite the Python community's insistence on using snake or reptile puns for their various packages and libraries, the language was not, in fact, named after the large constrictor populating various jungles all over the world. In fact, the Python appellation derives from the other famous Python: Monty. As in, Monty Python's Flying Circus, the cult classic comedy troupe from Britain from a few decades ago. It is the height of Python culture to use quotes and examples from any of the movies or episodes in any sample code, and good Pythonistas will never pass up the opportunity to toss a quip or two in passing. The Web site even insists on such.

You've been warned.

Getting Started

Getting started with Python is much easier than it might seem at first, for two basic reasons. One, a version of Python is available for install right out of the box with the Visual Studio installer. This is a packaged version of Python and environment management tools called Anaconda, and I'll talk about what that means and implies in a second. The second reason is that now, thanks to the Windows Subsystem for Linux (WSL), Python is just a simple bash-shell command-line command away, using the package manager of your choice. For example, if you've installed Ubuntu into the WSL environment, Python is easily obtained by opening an Ubuntu bash-shell prompt and typing "sudo apt-get install python3." Additionally, as of May, if you're running Windows 10, Python is available in the Windows Store; see bit.ly/2JMxC3A for details.

Numerous other options are also available, of course, if neither of these strikes your fancy. The Python Web site has a Windows-based installer available, for those who prefer their Python installation to come directly from the source. Or, for those who really want to get the source and nothing but the source, Python is (as most languages now are nowadays) an open source project hosted on GitHub and can be built from scratch. While either of these might have been the preferred approach a decade ago, the relatively recent success of bundling package managers as a core part of the language (starting with Ruby and gems, then migrating into Node and npm, not to mention Java and Maven, .NET and NuGet, and more) means that a modern Python installation needs to be accompanied by a package management tool, which in the core Python world is called "pip." As interest in Python has continued to grow, other deployment systems have emerged, as well, including the aforementioned Anaconda and its "conda" utility. There's no real need to stress when deciding between these, as they both get you to the same place. For the most part, if you use the Anaconda installation that comes with Visual Studio, you'll use conda, and if you choose the installer from the Python Web site, you'll use pip.

The other factor to be aware of is that Python comes in two sets of two major flavors. The first is that Python is available in both 64-bit and 32-bit versions; the differences in this case are pretty obvious. The second issue is that Python is officially making a rather tricky transition, that of moving from Python 2 to Python 3. For reasons that are largely unimportant to us, Guido von Rossum, the creator of Python, decided he had some breaking changes he needed to make to the language, and officially bumped the version number up when he implemented them. It was commonly expected that everybody would transition fairly quickly, but … that hasn't happened. Or rather, it's still happening. I'll be using Python 3 for the duration of this

series, as any new Python projects should start with Python3, but you may run across the odd package or library that still requires Python 2.

Once Python is installed, it's time to try it out. If you've chosen to install Anaconda, fire up the installed Start Menu item that reads "Anaconda Prompt." (Visual Studio 2019 actually installs this as "Python Prompt.") This brings up a command prompt that has the environment set for Anaconda Python, and it will look a little odd, as it will have something like "(base)" in front of the normal directory-based prompt ("C:\Users\Ted>" on my machine). This is because Anaconda is managing different "environments" for you, something I'll talk more about in a later column. For now, just type "python" to bring up the Python Read-Evaluate-Print Loop (REPL), the interactive environment, which will give you a ">>>" prompt after printing a version banner.

At this point, it's time to honor the Gods of Computer Science, by offering up the customary greeting:

```
print("Hello, Python world")
```

If you type this into the REPL, it will immediately print the greeting and provide another ">>>" prompt; if you wish, you can put this into a file (hello.py) and run it from the Anaconda prompt ("python hello.py"). Either way, you will have done your duty, and you can now list "Python programmer" on your resume.

If you want the message to be captured into a variable (perhaps for later reuse), you'd write it like so:

```
message = "Hello again, "
message += "Python world"
print(message)
```

Notice that the local variable, message, doesn't need to be declared before first use, and there's no "declaring keyword" like JavaScript's "var" (or "let" or "const"). Python supports strings, obviously, as well as Booleans, numeric values, lists, tuples, sets (lists that disallow duplication) and dictionaries (lists of tuples, or key-value pairs if you prefer) as built-in types. (You'll see in future columns how to build out custom object types, and explore what each of these primitive types means in more detail as we go.) Variables are themselves entirely untyped, which means your local variable can be assigned any kind of value at any time, like so:

```
# By the way, this is a comment
message = "Hello again, "
message += "Python world"
print(message)

message = 5
print(message)
```

However, this also brings up a critical point to take note of: Whenever a variable is introduced for the first time, it must have a value assigned to it, or Python will think you're trying to reference a previously defined variable. This is a small price to pay, though, considering that you can always assign zero or empty strings or whatever seems reasonable as a default value.

Last quick-hit note: As the first line of the example implies, the hash symbol (more properly known as the "octothorpe" to those who care about such things) is the end-of-line comment character in Python. Unlike many other languages, however, Python doesn't have a multi-line comment symbol, which means that commenting out large sections of code is a little less simple. However, this is entirely in keeping with the driving mantra of Python: "There's only one way to do it"; in this case, there's only one way to comment.

Executing Python

You can run this code in a variety of ways.

First, from a Windows command line, assuming Python is on the PATH, scripts can be executed by simply passing the name of the Python file to the interpreter, a la "python hello.py." This is likely to be the means for running Python applications for at least half of the Python applications you write, particularly if you use the tool to build graphs or run Web servers.

An interesting variation on this theme, though, just for Windows, is that if you use the Windows-based installers to put Python on your machine (as opposed to building from source), Python will register itself in the Windows Registry, and claim the ".py" file extension. Additionally, if ".PY" is appended to the "PATHEXT" environment variable in a Windows command prompt, you'll be allowed to run Python scripts as if they were directly executable (because they will be—this environment variable contains the extensions of all files that are intended to be directly executable, such as COM, EXE, BAT, CMD and others). This particular behavior is entirely specific to Windows, mind you, and not restricted to Python whatsoever—any file extension can be registered in the PATHEXT environment variable and be "directly executable," so long as the application-to-file-extension registration is set up in the Registry.

Variables are themselves entirely untyped, which means your local variable can be assigned any kind of value at any time.

On a Unix system, however, you can always set up a "she-bang" line at the top of the script; for those unfamiliar with this aspect of Unix behavior, when a script is "directly executed" (such as by typing "hello.py" at the command-line), Unix will examine the first line of the script, and if it reads something like "#!/usr/bin/env python3," the OS will assume that it's a command line to execute in order to run this particular script.

Wrapping Up

This is obviously just the first article of several on Python, so of course there are a few more topics to cover before you really get to understand Python and know how to use it. In the next piece, I'll cover Python's flow-control constructs, which will also bring me face-to-face with the next-most-interesting facet of Python: significant whitespace. Happy coding! ■

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

Thanks to the following technical expert for reviewing this article: *Harry Pierson*

Do It All with F# on .NET Core

Phillip Carter

F# is the functional programming language for .NET (bit.ly/2y4PeQG). It's cross-platform and, like all of .NET, it's open source (github.com/dotnet/fsharp). At Microsoft, we're huge fans of F# because it brings functional programming to .NET. For those who aren't familiar with the concept, the functional programming paradigm is one that emphasizes certain approaches:

- Functions as the primary constructs used to operate on data
- Expressions instead of statements
- Immutable values over variables
- Declarative programming over imperative programming

This means that F# brings some great features to .NET:

- Functions that are first-class (they can be passed as values to and returned from other functions)
- Lightweight syntax that emphasizes expressions and values, not statements
- Built-in immutability and non-null types
- Rich data types and advanced pattern matching techniques

Typical F# code often ends up looking like what's shown in **Figure 1**.

This article discusses:

- Getting started with F# using the .NET CLI
- Building a Web app with F#
- Using Span<'T> to enhance performance on .NET Core

Technologies discussed:

F#, .NET Core, Giraffe, ASP.NET Core

Code download available at:

bit.ly/2Z21yNq

Beyond these core features, F# can also interop with the entirety of .NET, and has full support for objects, interfaces and so forth. More advanced F# programming techniques often involve subtly combining object-oriented (OO) features with functional code, but without sacrificing the functional programming paradigm.

In addition, F# has lots of unique features that people love, such as Computation Expressions, Units of Measure, and powerful types such as Records and Discriminated Unions. See the F# Language Reference at bit.ly/2JSnipy for more information. F# also encourages a style of programming that tends toward safety and correctness: Many F# developers have turned to it after significant experience with other languages that don't emphasize safety and correctness as much. It has influenced a lot of the recent work that went into C#, such as async, tuples, pattern matching, and the forthcoming nullable reference types feature set.

> At Microsoft, we're huge fans of F# because it brings functional programming to .NET.

F# also has a vibrant community that loves to push the boundaries of .NET and create incredible open source components. The community is highly innovative and incredibly valuable to .NET, pioneering UI libraries, data processing libraries, testing methodologies, Web services and more for .NET!

The abundance of features and the vibrant community of enthusiastic developers have led many people to dive into F# both

LIVE! 360 AGENDA-AT-A-GLANCE

LIVE! 360 — TECH EVENTS WITH PERSPECTIVE

live360event

Visual Studio LIVE!

DevOps in the Spotlight | Database Development | Developing New Experiences | Delivery and Deployment | .NET Core and More | Full Stack Web Development

SQL Server LIVE!

Business Intelligence | SQL Server Administration & Maintenance | SQL Server Features & Components | SQL Server for Developers | SQL Server Performance Tuning and Optimization

Client and Endpoint Management | Powe and D

START TIME	END TIME	Visual Studio Live! Full Day Hands-On Labs: Sunday, November 17, 2019	SQL Server Live! Full Day Hands-On Lab: Sunday, November 17, 2019	TechMe
7:15 AM	9:00 AM			Registrat
9:00 AM	6:00 PM	**VSS01** Hands-On Lab: Xamarin—Beyond the Basics - *Marcel de Vries & Roy Cornelissen* **VSS02** Hands-on-Lab: Busy Developer's Guide to React - *Ted Neward* **VSS03** Hands-On Lab: Building Modern ASP.NET Core Web Apps the Right Way with Azure DevOps (Day 1) - *Philip Japikse & Brian Randell*	**SQS01** Hands-On Lab: The A to Z of Cloud-based SQL Server Solutions - *Allan Hirt*	**TMS01** Hands-On Lab: Before, Now Learn it H
2:00 PM	7:00 PM			Pre-Conference Regis

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, November 18, 2019	SQL Server Live! Pre-Conference Workshops: Monday, November 18, 2019	TechMento
7:00 AM	8:30 AM			Registra
8:30 AM	5:30 PM	**VSM01** Workshop: Developing Modern Web Apps with Azure - *Eric D. Boyd* **VSM02** Workshop: A Tour of Visual Studio 2019 - *Jason Bock* **VSS03** Hands-On Lab: Building Modern ASP.NET Core Web Apps the Right Way with Azure DevOps (Day 2) - *Philip Japikse & Brian Randell*	**SQM01** Workshop: Advanced Data Protection — Security and Privacy in SQL Server - *Thomas LaRock & Karen Lopez* **SQM02** Workshop: SQL Server 2019 - *Leonard Lobel*	**TMM01** Workshop: A A Demo-Filled Works
6:30 PM	8:00 PM			Dine-A-Round

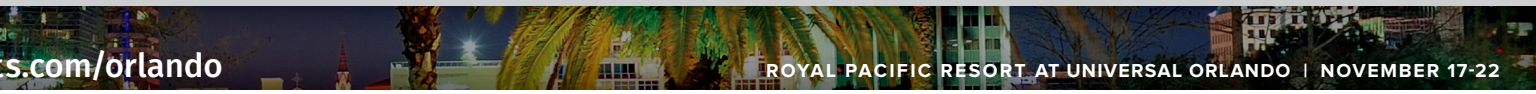
START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, November 19, 2019	SQL Server Live! Day 1: Tuesday, November 19, 2019	
7:00 AM	8:00 AM			Registra
8:00 AM	9:00 AM	**VISUAL STUDIO LIVE! KEYNOTE:** To Be Announced *Chloe Condon, Senior Cloud Advocate, Microsoft*	**SQL SERVER LIVE! KEYNOTE:** To Be Announced	*Chris Jackso*
9:15 AM	10:30 AM	**VST01** Moving to ASP.NET 2.X—Core Hands - *Philip Japikse* **VST02** .NET Developer's Introduction to Unity for 3D Apps & Games - *Nick Landry* **VST03** How Microsoft Does DevOps - *Paul Hacker* **VST04** What's New in .NET Core 3.0 - *Jason Bock*	**SQT01** AWS Versus Azure: Data Services Comparison - *Thomas LaRock* **SQT02** Availability Fundamentals for SQL Server - *Allan Hirt* **SQT03** Power BI: What Have You Done For Me Lately - *Andrew Brust*	**TMT01** PowerShell T Techniques Basics for IT Pr - *John O'Neill S*
10:30 AM	11:00 AM			Networkin
11:00 AM	12:00 PM			LIVE! 360 KEYNOTE: The Power of Real World DevOps - *Jessica*
12:00 PM	12:45 PM			
12:45 PM	1:30 PM			De
1:30 PM	1:50 PM	**VST05** Fast Focus: Hybrid Web Frameworks - *Allen Conway* **VST06** Fast Focus: What is Infrastructure as Code - *Paul Hacker* **VST07** Fast Focus: What's New in EF Core 3 - *Jim Wooley*	**SQT04** Fast Focus: Automation for the DBA: Embrace Your Inner Sloth - *William Durkin* **SQT05** Fast Focus: Who's Tinkling in Your Data Lake? - *Karen Lopez*	**TMT04** Fast Focus: T Cmdlets for Managing
2:00 PM	2:20 PM	**VST08** Fast Focus: Progressive Web Apps (PWA) Empowering Cloud-based Web Applications for Devices - *Andreas Erben* **VST09** Fast Focus: What is WSL and Why Do I Care? - *Brian Randell* **VST10** Fast Focus: Git Basics - *Mickey Gousset*	**SQT06** Fast Focus: From Adaptive to Intelligent: Query Processing in SQL 2019 - *Hugo Kornelis* **SQT07** Fast Focus: Performance Tuning Without Changing Code - *Thomas LaRock*	**TMT06** Fast Focus: I - *Jeff*
2:20 PM	2:45 PM			Networkin
2:45 PM	4:00 PM	**VST11** What's New in .NET Core 3 For Web Developers - *Philip Japikse* **VST12** Blazing the Web—Building Web Applications in C# - *Jason Bock* **VST13** Cosmos DB Applications: Part I - *Leonard Lobel* **VST14** To Be Announced	**SQT08** SQL Server 2019 Deep Dive - *Scott Klein* **SQT09** SQL Server In-Memory Database Objects - *Denny Cherry* **SQT10** What's New in SQL Server 2019 for Analytics - *Thomas LeBlanc*	**TMT08** An Inclusive Look Phishing - *Roger G*
4:15 PM	5:30 PM	**VST15** Getting Pushy with SignalR and Reactive Extensions - *Jim Wooley* **VST16** When "We are down" is Not Good Enough. Site Reliability Engineering (SRE) in Azure - *René van Osnabrugge* **VST17** Cosmos DB Applications: Part II - *Leonard Lobel* **VST18** What's New in C#8 - *Jason Bock*	**SQT11** Microsoft's New Database Experimentation Assistant (DEA) - *Mindy Curnutt* **SQT12** Common Troubleshooting Techniques for Availability Groups and Failover Cluster Instances - *Allan Hirt* **SQT13** Power BI implementation guidance: from Pro to Premium and Desktop to Embedded - *Thomas LeBlanc*	**TMT11** Fast-Track Your W Rockstar Status - *John O Cristal Kawuli*
5:30 PM	7:30 PM			Exh

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, November 20, 2019	SQL Server Live! Day 2: Wednesday, November 20, 2019	T
7:30 AM	8:00 AM			Registra
8:00 AM	9:15 AM	**VSW01** Migrating from AngularJS to Angular + TypeScript - *Allen Conway* **VSW02** AR/VR/Mixed Reality and HoloLens—Making it Real and Useful - *Andreas Erben* **VSW03** Scrum Under a Waterfall - *Benjamin Day* **VSW04** Getting Started with Entity Framework Core - *Jim Wooley*	**SQW01** An Introduction to Spatial Data in SQL Server - *Mindy Curnutt* **SQW02** It's Broken, Now What?! Practical Problem Solving - *William Durkin* **SQW03** Getting Started with Apache Spark - *Kevin Feasel*	**TMW01** 10 Ways to Hack - *Roger Grimes*
9:30 AM	10:45 AM	**VSW05** TypeScript: Moving Beyond the Basics - *Allen Conway* **VSW06** Busy Developer's Guide to Flutter - *Ted Neward* **VSW07** Get Comfortable with .NET Core and the CLI - *Jeremy Clark* **VSW08** To Be Announced	**SQW04** Blockchain for the DBA & Data Professional - *Karen Lopez* **SQW05** SQL Server Open Query Store - *William Durkin* **SQW06** PolyBase in Action - *Kevin Feasel*	**TMW04** Deploying and Office 365 ProPlus in a M - *Ronnie Peders*
10:45 AM	11:30 AM			Networkin
11:30 AM	12:30 PM			Live!
12:30 PM	1:30 PM			
1:30 PM	2:00 PM			De
2:00 PM	3:15 PM	**VSW09** Extending ASP.NET Core Identity to Suit Your Needs - *Jeremy Sinclair* **VSW10** (WPF + WinForms) * .NET Core = Modern Desktop - *Oren Novotny* **VSW11** Feature Flags for Better DevOps - *Mickey Gousset* **VSW12** Diving Deep into Entity Framework Core 2.x - *Philip Japikse*	**SQW07** SQL Data Discovery and Classification - *Karen Lopez* **SQW08** Things You Should Never Do In Microsoft SQL Server - *Denny Cherry* **SQW09** Deep Dive into Power BI Administration and Security Internals - *Ginger Grant*	**TMW07** Surviving a Ra Attack Using Azure Site Re Fest) - *Emile Cabot & Da*
3:15 PM	4:00 PM			Networking Break • Vis
4:00 PM	5:15 PM	**VSW13** Introduction to Webpack - *Chris Klug* **VSW14** Make Your WPF Applications Shine with Fluent Design - *Jeremy Sinclair* **VSW15** DevOps for Desktop Apps - *Oren Novotny* **VSW16** Run Faster: Parallel Programming in C# - *Jeremy Clark*	**SQW10** Migrating SSIS Workloads to Azure Data Factory - *Joshua Luedeman* **SQW11** Improve Your Database Performance in Seven Simple Steps - *Hugo Kornelis* **SQW12** Performance Tuning Power BI - *Ginger Grant*	**TMW10** Driving Adopt Modern Workplace - *Er*
7:30 PM	9:00 PM			Live! 360

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, November 21, 2019	SQL Server Live! Day 3: Thursday, November 21, 2019	
7:30 AM	8:00 AM			Registra
8:00 AM	9:15 AM	**VSH01** Advanced Fiddler Techniques - *Robert Boedigheimer* **VSH02** Release Management Strategies for Xamarin Developers - *Matthew Soucoup* **VSH03** DevOps ICU: Improving Software Dev Results by (Correctly) Integrating UX - *Debbie Levitt* **VSH04** Async internals in .NET - *Adam Furmanek*	**SQH01** Deep Dive into Adaptive Query Processing - *Hugo Kornelis* **SQH02** Using Modular Scripts to Perform SQL Compliance Audits in Seconds - *Chris Bell* **SQH03** DataOps and Business Intelligence Testing Approaches - *Jen Stirrup*	**TMH01** Build Your Azure like a Pro - *Aleksandar*
9:30 AM	10:45 AM	**VSH05** Improving Web Performance - *Robert Boedigheimer* **VSH06** Cross-Platform Development with Xamarin, Blazor, C#, and CSLA .NET - *Rockford Lhotka* **VSH07** DevInSecOps - *Doyle M. Turner* **VSH08** Internals of Exceptions - *Adam Furmanek*	**SQH04** Data Security and Privacy Techniques for Modern Databases - *Thomas LaRock* **SQH05** Building Your First SQL Server Container Lab in Docker - *Chris Bell* **SQH06** To Be Announced	**TMH04** How to Get Yo Services Secured with I Azure AD - *Peter Da*
11:00 AM	12:00 PM	**VISUAL STUDIO LIVE! PANEL DISCUSSION: Is There Such A Thing As A "Full Stack" Developer in 2019?** *Brian Randell (moderator), Adam Furmanek, Oren Novotny, Matthew Soucoup, & Alex Thissen*	**SQL SERVER LIVE! PANEL DISCUSSION: SQL Server is Dead, Long Live SQL Server!** *Thomas LaRock (moderator), Karen Lopez, Bradley Ball, Jen Stirrup, Joseph D'Antoni*	*Moderated*
12:00 PM	1:00 PM			
1:00 PM	2:15 PM	**VSH09** To Be Announced **VSH10** Place Your Bets: What Will Be the Next Breakthrough Dev Platform? - *Billy Hollis* **VSH11** Growing your DevOps mindset - *René van Osnabrugge* **VSH12** From .NET to .NET Core for Cloud Solutions - *Alex Thissen*	**SQH07** Machine Learning with R in Azure SQL Database - *Bradley Ball* **SQH08** Building a Better Data Solution: Microsoft SQL Server and Azure Data Services - *Joseph D'Antoni* **SQH09** Data Preparation: A framework with Power BI, Power Query, and Power BI Dataflows - *Jen Stirrup*	**TMH07** Building Real W in Azure - *Dave Ka*
2:30 PM	3:45 PM	**VSH13** To Be Announced **VSH14** The Most Important Lessons Learned in Forty Years of Developing Software - *Billy Hollis* **VSH15** (IAM) Secured - *Doyle M. Turner* **VSH16** Building Real World Production—Ready Web API's with .NET Core - *Alex Thissen*	**SQH10** TimescaleDB on Azure Database for PostgreSQL - *Bradley Ball* **SQH11** Containers, Pods, and Databases—Learning About the Future of Infrastructure - *Joseph D'Antoni* **SQH12** Getting Started with Power BI Report Server - *Joshua Luedeman*	**TMH10** Becoming a C Rockstar - *Cristal K*
4:00 PM	5:00 PM			**NEXT? LIVE! 360 NETWORKING EVENT** - *Bradley Ball, Andrew Brust, Be Thomas Sami Laiho, LaRock, Vishwas Lele, Rockford Lhotka, Karen Lop*

START TIME	END TIME	Visual Studio Live! Post-Conference Workshops: Friday, November 22, 2019	SQL Server Live! Post-Conference Workshops: Friday, November 22, 2019	TechMent
7:30 AM	8:00 AM			Registra
8:00 AM	5:00 PM	**VSF01** Workshop: Design an App UX in a Day - *Billy Hollis* **VSF02** Workshop: Web Development in 2019 - *Chris Klug*	**SQF01** Workshop: Azure SQL Data Warehouse: A Full Day of the Transformative New NDA Features - *Bradley Ball, Joshua Luedeman, & Gareth Swanepoel* **SQF02** Workshop: SQL Server High Performance Development - *Joseph D'Antoni*	**TMF01** Workshop: M Windows Server

Speakers and sessions subject to change



...s.com/orlando

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO | NOVEMBER 17-22

TECHMENTOR

...rShell ...evOps	Infrastructure	Soft Skills for IT Pros	Security	Azure (Public/ Hybrid)	Office 365 for the IT Pro

...ntor Full Day Hands-On Labs: Sunday, November 17, 2019

...tion • Coffee and Morning Pastries

...ou Got the Azure Workshops ...ands-on!! - *Peter De Tender*	**TMS02** Hands-On Lab: Rock JEA (Just Enough Administration) the Easy Way with Windows Admin Center - *John O'Neill Sr.*

...tration - Royal Pacific Resort Conference Center

...or Pre-Conference Workshops: Monday, November 18, 2019

...tion • Coffee and Morning Pastries

...zure from IAAS to PAAS— ...shop - *Koenraad Haedens*	**TMM02** Workshop: PowerShell Remoting Deep Dive - *Jeffery Hicks*

...d Dinner @ Universal CityWalk - 6:30pm

TechMentor Day 1: Tuesday, November 19, 2019

...tion • Coffee and Morning Pastries

TECHMENTOR KEYNOTE: To Be Announced
...n, Principal Architect, Windows Customer Experience Engineering, Microsoft

...ools and ...os, Not Devs ...r.	**TMT02** Protecting Your Corporate Resources Using Azure Identity - *Peter De Tender*	**TMT03** Zero Trust: Trust Identities, Not Your Network - *Ricky Pullan & Swetha Rai*

...g Break • Visit the EXPO - *Pacifica 7*

...Deen, Azure Avenger, Microsoft & Abel Wang, Senior Cloud Developer Advocate, Microsoft

Lunch • Visit the EXPO

...ssert Break • Visit the EXPO

...e 10 Handiest PowerShell ... Hyper-V - *John O'Neill Sr.*	**TMT05** Fast Focus: Rock Configuration Manager in 20 Minutes - *Emile Cabot*	
...PowerShell Fast & Furious ...ery Hicks	**TMT07** Fast Focus: Be a Better Speaker—How to Create Presentations that Inspire and Make People Care - *Erwin Derksen*	

...g Break • Visit the EXPO - Pacifica 7

...t Fighting ...imes	**TMT09** Troubleshooting the Modern Managed Client - *Ronnie Pedersen*	**TMT10** Azure Windows Virtual Desktop – Deploy and Configure in 50 Minutes - Koenraad Haedens
...Way to IT Pro ...'Neill Sr. & ...	**TMT12** Azure Security at Your Service (Learn Azure Security Center and Azure Sentinel) - *Peter De Tender*	**TMT13** Secure Remote Management with Just Enough Administration - *Jeffery Hicks*

...hibitor Reception - Pacifica 7

...echMentor Day 2: Wednesday, November 20, 2019

...tion • Coffee and Morning Pastries

...You Via Email ...	**TMW02** How YOU Deploy Office ProPlus / 2019 in the Enterprise Successfully - *Erwin Derksen*	**TMW03** Windows Powershell and Workflow: A Powerful Pairing! - *Michael Wiley*
... Managing ...odern World ...en	**TMW05** Managing Android and iOS in the Enterprise - *Peter Daalmans*	**TMW06** Become a PowerShell Debugging Ninja! - *Kirk Munro*

...g Break • Visit the EXPO - *Pacifica 7*

...360 Keynote: To Be Announced

... Birds-of-a-Feather Lunch

...ssert Break • Visit the EXPO

...nsomware ...covery (Demo ...ve Kawula	**TMW08** Everything You Need to Know to Manage Modern IT Apps Using Intune - *Peter Daalmans*	**TMW09** Creating and Sharing Awesome PowerShell Modules: A No Nonsense Guide - *Kirk Munro*
...tion of the ...mile Cabot	**TMW11** 5 Crucial Mistakes Made in IT-Projects (and How to Avoid Them) - *Erwin Derksen*	**TMW12** Life Beyond Functions: Creating Powershell Script Cmdlets (aka Advanced Functions) - *Michael Wiley*

...it the EXPO • Expo Raffle @ 3:30 p.m. - Pacifica 7

...0 Dessert Luau - Wantilan Pavilion

TechMentor Day 3: Thursday, November 21, 2019

...tion • Coffee and Morning Pastries

...Infrastructure ...r Nikolic	**TMH02** Imposter Syndrome: Overcoming Self-Doubt in Success - *Heather Downing*	**TMH03** Windows Autopilot Deep Dive - *Petri Paavola*
...our Cloud ...ntune and ...almans	**TMH05** Best Tips to Make the Life of SCCM Admins Easier! - *Panu Saukko*	**TMH06** What I Learned When I Moved my Operations to the Cloud - *Sami Laiho*

TECHMENTOR PANEL DISCUSSION: The Future of IT
...by Sami Laiho and Dave Kawula; Emile Cabot, Petri Paavola, & Panu Saukko

...Lunch - *Oceana Ballroom*

...World Labs ...wula	**TMH08** Implementing Proactive Security in the Cloud - *Sami Laiho*	**TMH09** How to Use PowerShell to Become a Windows Management SuperHero—2019 Edition - *Petri Paavola*
...ommunity ...awula	**TMH11** Managing Azure VMs with Windows Admin Center - *Aleksandar Nikolic*	**TMH12** Ready, Set, Go: Start Managing Your Windows 10 Devices with Intune - *Panu Saukko*

...n Curry, Peter Daalmans, Joseph D'Antoni, Marcel de Vries, Billy Hollis, Dave Kawula, Raj Krishnan, ...ez, Matthew McDermott, Agnes Molnar, Brian Randell, Panu Saukko, Jen Stirrup, & Alex Thissen

...or Post-Conference Workshops: Friday, November 22, 2019

...tion • Coffee and Morning Pastries

...igrating and Upgrading to ... 2019 - *Dave Kawula*	**TMF02** Workshop: How to Build Intune Managed Windows 10 with the Best User Experience - *Petri Paavola*

Artificial Intelligence LIVE!

AI App Develop-ment	Analytics	Bots	Data Science and ML	Internet of Things

AI Live! Full Day Hands-On Lab: Sun, Nov 17

AIS01 Hands-On Lab: Build Your Own A.I. Powered Robot - *Henk Boelman*	

AI Live! Pre-Conference Workshop: Mon, Nov 18

AIM01 Workshop: Workshop: Cloud Scale Machine Learning - *Seth Juarez & Cassie Sifjander*	**AIM02** Workshop: Big Data and Machine Learning with Hadoop, Spark, and SQL Server 2019 - *Andrew Brust*

Artificial Intelligence Live! Day 1: Tues, Nov 19

ARTIFICIAL INTELLIGENCE LIVE! KEYNOTE:
From Zero to AI Hero— Automatically Generate ML Models Using Azure Machine Learning Service, Automated ML - *Sujatha Sagiraju, Group Program Manager, Azure Cloud & AI Group, Microsoft*

AIT01 Make Your App See, Hear and Think with Cognitive Services - *Roy Cornelissen*	**AIT02** Data Insights with End-2-End Azure Analytics - *Scott Klein*

Artificial Intelligence Live! Day 2: Wed, Nov 20

AIT03 Fast Focus: Personal Assistants vs. Bots - *Brian Randell*	**AIT04** Fast Focus: The Sunny Side of AI - *Henk Boelman*
AIT05 Fast Focus: AR vs. VR vs. MR —What Does it All Mean? - *Nick Landry*	**AIT06** Fast Focus: AutoML Crash Course - *Andrew Brust*
AIW01 An Introduction to HoloLens & Mixed Reality for the Enterprise - *Nick Landry*	**AIW02** Machine Learning 101 for Developers - *Jen Underwood*
AIW03 Building a Holographic AI Assistant with ASP.NET Bots, Natural Language, & Mixed Reality - *Nick Landry*	**AIW04** Getting Started with Azure Machine Learning Services - *Henk Boelman*
AIT07 Did You Know Cognitive Services Could Do This? A Computer Vision Quest - *Andreas Erben*	**AIT08** AI and Analytics with Apache Spark on Azure Databricks - Andrew Brust
AIT09 Bot Building 101 with the Microsoft Bot Framework - *Brian Randell*	**AIT10** Diving Into Deep Learning with Databricks - *Ginger Grant*
AIW05 Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*	**AIW06** DevOps for Artificial Intelligence, the Road to Production - *Henk Boelman*
AIW07 The Edge of Tomorrow, Use Visual Studio Code and Raspberry Pi to Create IoT Edge Solutions - *Nicholas McCollum*	**AIW08** Visualization Best Practices for Machine Learning Applications - *Jen Underwood*

ARTIFICIAL INTELLIGENCE LIVE! PANEL DISCUSSION: AI—For Data Scientists or Developers? - *Andrew Brust (moderator), Kevin Feasel, Raj Krishnan, Heather Downing*

Artificial Intelligence Live! Day 3: Thurs, Nov 21

AIH01 Machine Learning the .NET Way - *Bri Achtman*	**AIH02** Machine Learning with Power BI - *Raj Krishnan*
AIH03 Demystifying User Management for Voice Apps - *Heather Downing*	**AIH04** Launching A Data Science Project: Cleaning Is Half the Battle - *Kevin Feasel*
AIH06 Google vs. Alexa: Battle of the Bots - *Heather Downing*	**AIH06** Data Orchestration and Data Flow with Azure Data Factory - *Raj Krishnan*
AIH07 To Be Announced	**AIH08** The 8 "C's" of Artificial Intelligence – Being Successful with AI - *Jen Stirrup*

AI Live! Post-Conf. Workshop: Fri, Nov 22

AIF01 Workshop: From Business Intelligence to Business Analytics with the Microsoft Data Platform - *Jen Stirrup*	

Cloud & Containers LIVE!

Cloud-Native Development	Infrastructure	Cloud DevOps

Cloud & Containers Live! Full Day Hands-On Lab: Sun, Nov 17

CCS01 Hands-On Lab: Hands-On with Cloud-Native .NET Development - *Rockford Lhotka*	

Cloud & Containers Live! Pre-Conf. Workshops: Mon, Nov 18

CCM01 Workshop: Building, Running & Continuously Deploying Microservices with Docker Containers on Azure - *Marcel de Vries & René van Osnabrugge*	**CCM02** Workshop: Kubernetes Zero to Hero—Installation, Configuration, and Application Deployment - *Anthony Nocentino*

Cloud & Containers Live! Day 1: Tuesday, November 19, 2019

CLOUD & CONTAINERS LIVE! KEYNOTE: To Be Announced

CCT01 Serverless with Knative - *Mete Atamel*	**CCT02** Intro to Kubernetes - *Chris Kinsman*

Cloud & Containers Live! Day 2: Wed, Nov 20, 2019

CCT03 Fast Focus: Serverless in Azure 101 - *Eric D. Boyd*	**CCT04** Fast Focus: Containers on AWS - *Chris Kinsman*
CCT05 Fast Focus: Deploying Apps to Kubernetes Using Helm - *Rockford Lhotka*	**CCT06** Fast Focus: Building & Debugging Container-based Apps with VS2019 - *Benjamin Day*
CCT07 Building a Better Distributed Job Scheduler on Kubernetes - Chris Kinsman	**CCT08** Lock the Doors, Secure the Valuables, and Set the Alarm - Eric D. Boyd
CCT09 Microservices and Containers with Service Fabric and Azure Service Fabric Mesh - *Eric D. Boyd*	**CCT10** Practical Container Scenarios in Azure - *Anthony Nocentino*
CCW01 Azure App Service Security for ASP.NET Core MVC and WebAPI - *Benjamin Day*	**CCW02** To Be Announced
CCW03 Microservice Architecture - *Rockford Lhotka*	**CCW04** Automate Your Life with PowerShell on Lambda - *AM Grobelny & Steve Roberts*
CCW05 Architecting .NET Solutions in a Docker Ecosystem - *Alex Thissen*	**CCW06** To Be Announced
CCW07 Serverless .NET on AWS - *AM Grobelny*	**CCW08** To Be Announced

CLOUD & CONTAINERS LIVE! PANEL DISCUSSION: What the Cloud-Native Evolution Means to You - *Rockford Lhotka (moderator), Vishwas Lele, Marcel de Vries, Nick Pinheiro, & Jen Tong*

Cloud & Containers Live! Day 3: Thurs, Nov 21, 2019

CCH01 Busy Developer's Guide to the Clouds - *Ted Neward*	**CCH02** Kubernetes Runtime Security - *Jen Tong*
CCH03 Standup Comedy Hour—Why App Folks Love Infrastructure Folks? - *Vishwas Lele & Jack O'Connell*	**CCH04** Implementing Zero Downtime Application Deployments on Azure PaaS - *Marcel de Vries*
CCH05 Modernize Your App to be Delivered as a SaaS Service - *Nick Pinheiro*	**CCH06** Azure Cloud Transformation Done Right - *Marcel de Vries*
CCH07 Enable External Access to Your Custom Apps with Azure AD B2B - *Nick Pinheiro*	**CCH08** Running 30-Year-Old Software as a Cloud Native SaaS Solution with Docker and Kubernetes on Azure - *Roy Cornelissen*

Cloud & Containers Live! Post-Conf. Workshop: Fri, Nov 22

CCF01 Workshop: Kubernetes on Azure - *Vishwas Lele*	

Office & SharePoint LIVE!

Managing Office 365	SharePoint/ Office 365/ Cloud	High-Value SharePoint Workloads	Developing for Office/ SharePoint	Office 365 Identity and Security

O&SP Live! Full Day Hands-On Lab: Sun, Nov 17

O&SP Live! Pre-Conf. Workshop: Mon, Nov 18

OSM01 Workshop: Quick Wins to Jump Start Your SharePoint Implementation - *Stacy Deere-Strole*	

Office & SharePoint Live! Day 1: Tues, Nov 19

OFFICE & SHAREPOINT LIVE! KEYNOTE: To Be Announced

OST01 Mastering Modern Authentication and Authorization Techniques for SharePoint, Office 365 and Azure AD - *Eric Shupps*	**OST02** Understanding Microsoft Search—What You Want to Know - *Agnes Molnar*

Office & SharePoint Live! Day 2: Wed, Nov 20

OST03 Fast Focus: Personalizing Your Office 365 Tenant to Help with User Adoption - *Dan Usher*	
OST04 Fast Focus: Build a Bot for Microsoft Teams in 5 Minutes Using Microsoft Azure - *Noorez Khamis*	
OST05 To Be Announced	**OST06** Navigation: A Step Towards Success in SharePoint - *Stacy Deere-Strole*
OST07 Building Your First App with the Microsoft Graph (Office 365 APIs) - *Rob Windsor*	**OST08** Getting Started with Office 365 - *Dan Usher*
OSW01 Microsoft Flow for SharePoint Developers - *Rob Windsor*	**OSW02** Creating a Security Plan for Office 365—The Blue Pill Session - *Ben Curry*
OSW03 Taking User Profiles to New Heights! - *Stacy Deere-Strole*	**OSW04** Managing Your Microsoft Teams - *Dan Usher*
OSW05 An Introduction to Development with the SharePoint Framework (SPFx) - *Rob Windsor*	**OSW06** A Real World Guide to Building Highly Available Fault Tolerant SharePoint Farms - *Eric Shupps*
OSW07 Supercharge Your Business Productivity with Teams, Extensions, and SPFx - *Bill Ayers*	**OSW08** Content Services in Office 365 - *Agnes Molnar*

Office & SharePoint Live! Day 3: Thurs, Nov 21

OSH01 Developing SharePoint Framework Solutions for the Enterprise - *Eric Shupps*	**OSH02** Automating Business Process on Your Corporate Files Using OneDrive and Flow - *Noorez Khamis*
OSH03 Asynchronous JavaScript and TypeScript for SharePoint Developers - *Rob Windsor*	**OSH04** Search Like a Pro: Mythbusting the "Black Box" of Search - *Agnes Molnar*
OSH05 Introduction to Microsoft Teams Development - *Noorez Khamis*	**OSH06** Office 365 Data Security—The Red Pill Session - *Ben Curry*
OSH07 Turn Your Business Productivity Up To 11 with Actionable Messages and Adaptive Cards - *Bill Ayers*	**OSH08** Running an Modern Search Project in SharePoint Online - *Matthew McDermott*

OFFICE & SHAREPOINT LIVE! PANEL DISCUSSION: Privacy, Security, and Compliance, It's a "We" Problem - *Matthew McDermott (moderator), Bill Ayers, Ben Curry, Agnes Molnar, & Eric Shupps*

O&SP Live! Post-Conf. Workshop: Fri, Nov 22

OSF01 Workshop: Six Easy Pieces - Microsoft 365 Extensions that Bring Maximum Business Benefit with Minimum Effort - *Bill Ayers*	

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

6 Great Events, 1 Low Price!

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

Visual Studio LIVE!
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

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

SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

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

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

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

Artificial Intelligence LIVE!
AI FOR DEVELOPERS AND DATA SCIENTISTS

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

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

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

NEW! IN 2019

Cloud & Containers LIVE!
CLOUD-NATIVE, PAAS & SERVERLESS COMPUTING

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

REGISTER NOW

SAVE $400 WHEN YOU REGISTER BY SEPTEMBER 27*

Use Promo Code: MSDN

* Restrictions Apply.

CONNECT WITH LIVE! 360

twitter.com/live360
@live360
facebook.com
Search "Live 360"
instagram.com
@live360_events
linkedin.com
Join the "Live! 360" group!

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

HIGHLIGHTED KEYNOTES AND FEATURED SPEAKERS

Live! 360 brings experts from around the globe to connect with you in our Keynotes, Hands-On Labs, Workshops and Sessions. Check out a few of our highlighted Keynotes and featured speakers, bringing some of the best and brightest experts in the industry to our full conference agenda.

View the full list of speakers at live360events.com/orlando

Live! 360 Keynote: The Power of Real World DevOps

Jessica Deen, *Azure Avenger, Microsoft*

Abel Wang, *Senior Cloud Developer Advocate, Microsoft*

Artificial Intelligence Live! Keynote: From Zero to AI Hero—Automatically Generate ML Models Using Azure Machine Learning Service, Automated ML

Sujatha Sagiraju, *Group Program Manager, Azure Cloud & AI Group, Microsoft*

Visual Studio Live! Keynote: To Be Announced

Chloe Condon, *Senior Cloud Advocate, Microsoft*

Visual Studio LIVE!

Jason Bock
MVP (C#), Practice Lead, Magenic

Billy Hollis
XAML Slinger

Chris Klug
Senior Software Developer, Tretton37

Debbie Levitt
CEO, Ptype UX Agency

Jeremy Sinclair
Web Engineering Specialist III, Orrick

SQL Server LIVE!

Bradley Ball
Sr. Azure Engineer, Azure FastTrack CXP, Microsoft

Denny Cherry
Owner, Denny Cherry & Associates Consulting

Ginger Grant
Principal Data Geek, Desert Isle Group

Leonard Lobel
CTO, Sleek Technologies, Inc.

Jen Stirrup
Principal, Data Relish Ltd

TECHMENTOR

Heather Downing
Coder. Entrepreneur.

Jeffery Hicks
PowerShell Sensei

Dave Kawula
Principal Consultant, TriCon Elite Consulting

Petri Paavola
Yodamitti Oy

Swetha Rai
Program Manager II - Azure Active Directory, Microsoft

Artificial Intelligence LIVE!

Andreas Erben
CTO, Industrial Holographics

Raj Krishnan
Cloud Solution Architect, Microsoft

Nick Landry
Senior Software Engineer, Microsoft

Brian Randell
Partner, MCW Technologies

Jen Underwood
Founder, Impact Analytix, LLC

Office & SharePoint LIVE!

Stacy Deere-Strole
Solutions Architect, Focal Point Solutions LLC

Noorez Khamis
VP Technology, Creospark

Matthew McDermott
Principal Technical Marketing Engineer, Spanning Cloud Apps

Agnes Molnar
Founder, Managing Consultant, Search Explained

Eric Shupps
Founder & President, BinaryWave

Cloud & Containers LIVE!

Chris Kinsman
Chief Architect, PushSpring

Vishwas Lele
CTO, Applied Information Sciences

Rockford Lhotka
CTO, Magenic

Ted Neward
Principal, Neward & Associates, LLC

Jen Tong
Security Advocate, Google

live360events.com/orlando

REGISTER NOW

SAVE $400 WHEN YOU REGISTER BY SEPTEMBER 27*

Use Promo Code MSDN

*Restrictions apply. See website for details.

for fun and for their work. F# powers financial institutions across the world, eCommerce systems, scientific computing, organizations doing data science and machine learning, consultancies, and more. F# is also used quite a bit at Microsoft: As one of the primary languages used by Microsoft Research, it influenced and helped power the Q# development platform; parts of Azure and Office 365; and even the F# compiler and Visual Studio tools for F#! In short, it's used everywhere.

Interested? Great! In this article, I'll show you how to do some cool stuff with F# on .NET Core.

Overview

I'll start with the basics of F# on .NET Core and walk though progressively more advanced (and more interesting) capabilities, including how you can use the .NET CLI to create a console application and library project on any OS. Although simple and bare-bones, this toolset alone can be enough to develop an entire application when coupled with an editor like Visual Studio Code and the official F# plug-in, Ionide (ionide.io). You can also use Vim or Emacs if that's your thing.

Next, I'll give a brief overview of some technologies for building Web services. Each has some different tradeoffs that are worth mentioning. I'll then show you an example using one of these technologies.

Finally, I'll talk a bit about how you can leverage some of the high-performance constructs in .NET Core, like Span<'T>, to cut down on allocations and really speed up the hot paths in your system.

Get Started with F# Using the .NET CLI

One of the easiest ways to get started with F# is to use the .NET CLI. First, ensure you've installed the latest .NET SDK.

Now let's create a few projects that are all connected to each other. Modern terminals support tab completion, so even though there's a bit to type, your environment should be able to complete much of it for you. To begin, I'll create a new solution:

```
dotnet new sln -o FSNetCore && cd FSNetCore
```

This will create a new directory called FSNetCore, create a solution file in that directory, and change directories to FSNetCore.

Next, I'll create a library project and hook it up to the solution file:

```
dotnet new lib -lang F# -o src/Library
dotnet sln add src/Library/Library.fsproj
```

I'll add a package to that library:

```
dotnet add src/Library/Library.fsproj package Newtonsoft.Json
```

This will add the Json.NET package to the project.

> The abundance of features and the vibrant community of enthusiastic developers have led many people to dive into F# both for fun and for their work.

I'll now change the Library.fs file in the library project to be the following:

```
module Library

open Newtonsoft.Json

let getJsonNetJson value =
  sprintf "I used to be %s but now I'm %s thanks to JSON.NET!" value
(JsonConvert.SerializeObject(value))
```

This is a module that contains a single F# function that uses JSON.NET to serialize a generic value into a JSON string with the rest of a message.

Next, I'll create a console app that consumes the library:

```
dotnet new console -lang F# -o src/App
dotnet add src/App/App.fsproj reference src/Library/Library.fsproj
dotnet sln add src/App/App.fsproj
```

I'll replace the Program.fs contents in the console app project with the following:

```
open System

[<EntryPoint>]
let main argv =
  printfn "Nice command-line arguments! Here's what JSON.NET has to say about them:"
  argv
  |> Array.map Library.getJsonNetJson
  |> Array.iter (printfn "%s")

  0 // Return an integer exit code
```

This will take each command-line argument, transform it into a string defined by the library function, and then iterate over those strings and print them out.

I can now run the project:

```
dotnet run -p src/App Hello World
```

This will print the following to the console:

```
Nice command-line arguments! Here's what JSON.NET has to say about them:

I used to be Hello but now I'm ""Hello"" thanks to JSON.NET!
I used to be World but now I'm ""World"" thanks to JSON.NET!
```

Pretty easy, right? Let's add a unit test and connect it to the solution and library:

Figure 1 **Typical F# Code**

```
// Group data with Records
type SuccessfulWithdrawal = {
  Amount: decimal
  Balance: decimal
}

type FailedWithdrawal = {
  Amount: decimal
  Balance: decimal
  IsOverdraft: bool
}

// Use discriminated unions to represent data of 1 or more forms
type WithdrawalResult =
  | Success of SuccessfulWithdrawal
  | InsufficientFunds of FailedWithdrawal
  | CardExpired of System.DateTime
  | UndisclosedFailure

let handleWithdrawal amount =
  // Define an inner function to hide it from callers
  // Returns a WithdrawalResult
  let withdrawMoney amount =
    callDatabaseWith amount // Let's assume this has been implemented :)

  let withdrawalResult = withdrawMoney amount

  // The F# compiler enforces accounting for each case!
  match w with
  | Success s -> printfn "Successfully withdrew %f" s.Amount
  | InsufficientFunds f -> printfn "Failed: balance is %f" f.Balance
  | CardExpired d -> printfn "Failed: card expired on %O" d
  | UndisclosedFailure -> printfn "Failed: unknown :("
```

Visual Studio LIVE!
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

CHICAGO

Navigate Today's Tech in the Windy City

OCTOBER 6-10, 2019
SWISSOTEL, CHICAGO, IL

#VSLive

INTENSE DEVELOPER TRAINING CONFERENCE

In-depth Technical Content On:

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

New This Year!

On-Demand Session Recordings Now Available

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

SAVE $300! When You Register by September 13

Your Adventure Starts Here!

EVENT PARTNER Microsoft

GOLD SPONSOR CSC corptax.com

SUPPORTED BY Visual Studio, msdn magazine, Visual Studio MAGAZINE

PRODUCED BY CONVERGE 360 An 1105 MEDIA company

vslive.com/chicago

AGENDA AT-A-GLANCE

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

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

Speakers and sessions subject to change

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

```
dotnet new xunit -lang F# -o tests/LibTests
dotnet add tests/LibTests/LibTests.fsproj reference src/Library/Library.fsproj
dotnet sln add tests/LibTests/LibTests.fsproj
```

Now I'll replace the Tests.fs contents in the test project with the following:

```
module Tests

open Xunit

[<Fact>]
let ``Test Hello`` () =
  let expected = """I used to be Hello but now I'm "Hello" thanks to JSON.NET!"""
  let actual = Library.getJsonNetJson "Hello"
  Assert.Equal(expected, actual)
```

This simple test just verifies that the output is correct. Note that the test name is enclosed by double-backticks to allow the use of a more natural name for the test. This is quite common in F# testing. Additionally, you use the triple-quoted string when you want to embed quoted strings in F#. Alternatively, you could use backslashes if you prefer.

Now I can run the test:

```
dotnet test
```

And the output verifies that it passes!

```
Starting test execution, please wait...

Test Run Successful.
Total tests: 1
Passed: 1
Total time: 4.9443 Seconds
```

Ta-da! With a minimal toolset, it's entirely possible to build a library that's unit-tested and a console app that runs that library code. This alone is enough to actually build a full solution, especially if you pair it with an F# code editor like Visual Studio Code and the Ionide plug-in. In fact, many professional F# developers use only this to do their daily work! You can find the full solution on GitHub at bit.ly/2Svquuc.

This is quite fun, but let's look at some material that's more interesting than a library project or a console app.

Build a Web App with F#

F# can be used for a lot more than just library projects and console apps. Among the most common solutions F# developers build are Web apps and services. There are three main options for doing this, and I'll briefly discuss each one:

Giraffe (bit.ly/2Z4zPeP) is best thought of as "functional programming bindings to ASP.NET Core." It's fundamentally a middleware library that exposes routes as F# functions. Giraffe is somewhat of a "bare bones" library that's relatively unopinionated about how you build your Web services or Web app. It offers some wonderful ways to compose routes using functional techniques and has some nice built-in functions to simplify things like working with JSON or XML, but how you compose your projects is entirely up to you. Lots of F# developers use Giraffe because of how flexible it is, and because it has such a rock-solid foundation as it uses ASP.NET Core under the covers.

Saturn (bit.ly/2YjgGsl) is sort of like "functional programming bindings to ASP.NET Core, but with batteries included." It uses a form of the MVC pattern, but done functionally rather than with object-oriented programming abstractions. It's built atop Giraffe and shares its core abstractions, but it gives a much more opinionated take on how to build Web apps with .NET Core. It also includes some .NET CLI command-line tools that offer database model generation, database migrations, and scaffolding of Web controllers and views. Saturn includes more built-in functions than Giraffe as a result of being more opinionated, but it does require that you buy into the approach it enforces.

Suave (bit.ly/2YmDRSJ) has been around for a lot longer than Giraffe and Saturn. It was the primary influence for Giraffe because it pioneered the programming model that both use in F# for Web services. A key difference is that Suave has its own OWIN-compliant Web server that it uses, rather than sitting atop ASP.NET Core. This Web server is highly portable, as it can be embedded into low-powered devices via Mono. The programming model for Suave is slightly simpler than Giraffe due to not needing to work with ASP.NET Core abstractions.

With a minimal toolset, it's entirely possible to build a library that's unit-tested and a console app that runs that library code.

I'll start by building a simple Web service with Giraffe. It's easy with the .NET CLI:

```
dotnet new -i "giraffe-template::*"
dotnet new giraffe -lang F# -V none -o GiraffeApp
cd GiraffeApp
```

Now I can build the application by running either build.bat or sh build.sh.

Now I'll run the project:

```
dotnet run -p src/GiraffeApp
```

I can then navigate to the route given by the template /api/hello, via localhost.

Navigating to https://localhost:5001/api/hello gives me:

```
{"text":"Hello world, from Giraffe!"}
```

Cool! Let's take a look at how this was generated. To do that, I'll open up the Program.fs file and note the webApp function:

```
let webApp =
  choose [
    subRoute "/api"
      (choose [
        GET >=> choose [
          route "/hello" >=> handleGetHello
        ]
      ])
    setStatusCode 404 >=> text "Not Found" ]
```

There's a bit going on here, and it's actually backed by some rather intricate functional programming concepts. But it's ultimately a domain-specific language (DSL), and you don't need to understand every little bit of how it works to use it. Here are the basics:

The webApp function is comprised of a function called choose. The choose function is executed by the ASP.NET Core runtime whenever someone makes a request to the server. It will look at the request and try to find a route that matches. If it can't, it will fall back to the 404 route defined at the bottom.

Because I have a subRoute defined, the choose function will know to crawl all of its child routes. The definition of what to crawl

is specified by an inner choose function and its list of routes. As you can see, within that list of routes, there's a route that specifies the "/hello" string as its route pattern.

This route is actually another F# function, called route. It takes a string parameter to specify the route pattern, and is then composed with the >=> operator. This is a way of saying, "this route corresponds to the function that follows it." This kind of composition is what's known as Kleisli composition, and although it's not essential to understand that theoretical underpinning to use the operator, it's worth knowing that it has a firm mathematical foundation. As I mentioned earlier in the article, F# developers lean toward correctness ... and what's more correct than a mathematical basis?

You'll notice that the function on the right-hand side of the >=> operator, handleGetHello, is defined elsewhere. Let's open up the file it's defined in, HttpHandlers.fs:

```
let handleGetHello =
  fun (next: HttpFunc) (ctx: HttpContext) ->
    task {
      let response = {
        Text = "Hello world, from Giraffe!"
      }
      return! json response next ctx
    }
```

Unlike "normal" F# functions, this handler function is actually defined as a lambda: It's a first-class function. Although this style isn't quite so common in F#, it's chosen because the two parameters that the lambda takes—next and ctx—are typically constructed and passed to the handler by the underlying ASP.NET Core runtime, not necessarily user code. From our perspective as programmers, we don't need to pass these around ourselves.

These parameters defined in the lambda function are abstractions that are defined in and used by the ASP.NET Core runtime itself. Once in the body of the lambda function, with these parameters, you can construct any object you'd like to serialize and send down the ASP.NET Core pipeline. The value called response is an instance of an F# record type that contains a single label, Text. Because Text is a string, it's given a string to serialize. The definition of this type resides in the Models.fs file. The function then returns a JSON-encoded representation of the response, with the next and ctx parameters.

Another way you can look at a Giraffe pipeline is with some small boilerplate at the top and bottom of a function, to conform to the ASP.NET Core pipeline abstraction and then anything you want in between:

```
let handlerName =
  fun (next: HttpFunc) (ctx: HttpContext) ->
    task {
      // Do anything you want here
      //
      // ... Well, anything within reason!
      //
      // Eventually, you'll construct a response value of some kind,
      // and you'll want to serialize it (as JSON, XML or whatever).
      //
      // Giraffe has multiple middleware-function utilities you can call.

      return! middleware-function response next ctx
    }
```

Although this may seem like a lot to learn up front, it's quite productive and easy to build Web apps and services with. To demonstrate this, I'll add a new handler function in the HttpHandlers.fs file that gives a greeting if you specify your name:

```
let handleGetHelloWithName (name: string) =
  fun (next: HttpFunc) (ctx: HttpContext) ->
    task {
      let response = {
        Text = sprintf "Hello, %s" name
       }
      return! json response next ctx
    }
```

As before, I set up this handler with the necessary boilerplate to conform to ASP.NET Core middleware. A key difference is that my handler function takes a string as input. I use the same response type as before.

Next, I'll add a new route in the Program.fs file, but because I want to specify some arbitrary string as input, I'll need to use something other than the route function. Giraffe defines the routef function exactly for this purpose:

```
let webApp =
  choose [
    subRoute "/api"
      (choose [
        GET >=> choose [
          route "/hello" >=> handleGetHello

          // New route function added here
          routef "/hello/%s" handleGetHelloWithName
        ]
      ])
    setStatusCode 404 >=> text "Not Found" ]
```

The routef function takes in two inputs:

- A format string that represents the route and its input (in this case, a string with %s)
- A handler function (that I defined earlier)

You'll note that I didn't supply the >=> operator here. This is because routef has two parameters: the string pattern (specified by an F# format string) and the handler that operates on types specified by the F# format string. This is in contrast with the route function, which only takes a string pattern as input. In this case, because I don't need to compose routef and my handler with anything else, I don't use the >=> operator to compose additional handlers. But if I wanted to do something like set a specific HTTP status code, I'd do that by composing with >=>.

> Lots of F# developers use Giraffe because of how flexible it is, and also because it has such a rock-solid foundation as it uses ASP.NET Core under the covers.

Now I can rebuild the app and navigate to https://localhost:5001/api/hello/phillip. When I do, I get:

```
{"text":"Hello, Phillip"}
```

Ta-da! Pretty easy, right? As with any library or framework, there are a few things to learn, but once you're comfortable with the abstractions it's incredibly easy to add routes and handlers that do what you need.

You can read more about how Giraffe works in its excellent documentation (bit.ly/2GqBVhT). And you'll find a runnable sample app that shows what I demonstrated at bit.ly/2Z21yNq.

Figure 2 **Benchmarking a Parsing Routine with and without Span<'T>**

```
open System
open BenchmarkDotNet.Attributes
open BenchmarkDotNet.Running

module Parsing =
  /// "123,456" --> (123, 456)
  let getNums (str: string) (delim: char) =
    let idx = str.IndexOf(delim)
    let first = Int32.Parse(str.Substring(0, idx))
    let second = Int32.Parse(str.Substring(idx + 1))
    first, second

  let getNumsFaster (str: string) (delim: char) =
    let sp = str.AsSpan()
    let idx = sp.IndexOf(delim)
    let first = Int32.Parse(sp.Slice(0, idx))
    let second = Int32.Parse(sp.Slice(idx + 1))
    struct(first, second)

[<MemoryDiagnoser>]
type ParsingBench() =
  let str = "123,456"
  let delim = ','

  [<Benchmark(Baseline=true)>]
  member __.GetNums() =
    Parsing.getNums str delim |> ignore

  [<Benchmark>]
  member __.GetNumsFaster() =
    Parsing.getNumsSpan str delim |> ignore

[<EntryPoint>]
let main _ =
  let summary = BenchmarkRunner.Run<ParsingBench>()
  printfn "%A" summary

  0 // Return an integer exit code
```

Go Faster

Now I'll diverge from the practical application of F# to delve into some performance characteristics.

When building something like a Web service that has high traffic, performance matters! Specifically, avoiding needless allocations for the GC to clean up tends to be one of the most impactful things you can do for long-running Web server processes.

This is where types like Span<'T> start to shine when you're using F# and .NET Core. A span is sort of like a window into a buffer of data that you can use to read and manipulate that data. Span<'T> imposes a variety of restrictions on how you can use it so that the runtime can guarantee that various performance enhancements will apply.

I'll demonstrate this with a sample (as seen in "All About Span: Exploring a new .NET Mainstay" by Steven Toub at msdn.com/magazine/mt814808). I'll use BenchmarkDotNet to measure the results.

First, I'll create a console app on .NET Core:

```
dotnet new console -lang F# -o Benchmark && cd Benchmark
dotnet add package benchmarkdotnet
```

Next, I'll modify it to benchmark a routine that has a typical implementation and an implementation that uses Span<'T>, as shown in **Figure 2**.

The module called Parsing contains two functions that split a string by a given delimiter, returning a tuple representing each half of the string. However, one called getNumsFaster uses both Span<'T> and struct tuples to eliminate allocations. As you'll see, the results are quite profound.

I'll run the benchmark to produce the results:

```
dotnet run -c release
```

This will produce results that can be shared as markdown, HTML or other formats.

> When building something like a Web service that has high traffic, performance matters!

I ran this benchmark on my laptop with the following hardware and runtime environment:

- BenchmarkDotNet v0.11.5
- macOS Mojave 10.14.5 (18F132) [Darwin 18.6.0]
- Intel Core i7-7700HQ CPU 2.80GHz (Kaby Lake), 1 CPU, 8 logical and 4 physical cores
- .NET Core SDK=3.0.100-preview5-011568 (64-bit)

The results are shown in **Figure 3**.

Impressive, right? The getNumsFaster routine not only allocated 0 additional bytes, it also ran 33 percent faster!

If you're still not convinced that this matters, imagine a scenario where you need to perform 100 transformations on that data, and it all had to happen on the hot path for a highly trafficked Web service. If that service saw requests on the order of millions per second, you'd be looking at a pretty severe performance problem if you're allocating in each transformation (or several of them). However, if you use types like Span<'T> and struct tuples, all of those allocations can often disappear. And, as the benchmark shows, it can also take significantly less wall-clock time to execute a given operation.

Wrapping up

As you can tell, there's quite a lot you can do with F# on .NET Core! It's very easy to get started, and easy to work your way into building Web applications, too. Moreover, the ability to use constructs like Span<'T> means that F# can be used for performance-sensitive work, as well.

F# is only getting better on .NET Core, and the community continues to grow. We'd love it if you'd join the community in building some of the next great things for F# and .NET! ■

Figure 3 **Benchmark Results**

Method	Mean	Error	StdDev	Ratio	Gen0	Gen1	Gen2	Allocated
GetNums	90.17 ns	0.6340 ns	0.5620 ns	1.00	0.5386	-	-	88 B
GetNumsFaster	60.01 ns	0.2480 ns	0.2071 ns	0.67	-	-	-	-

Phillip Carter *is a member of the .NET team at Microsoft. He works on the F# language and tools, F# documentation, the C# compiler and .NET project integration tooling for Visual Studio.*

Thanks to the following technical expert for reviewing this article:
Dustin Moris Gorski

Does your company create enterprise computing solutions for customers?

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

CHALLENGE

The Microsoft ecosystem is moving at a blistering pace

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

SOLUTION

Redmond Intelligence Research Reports

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

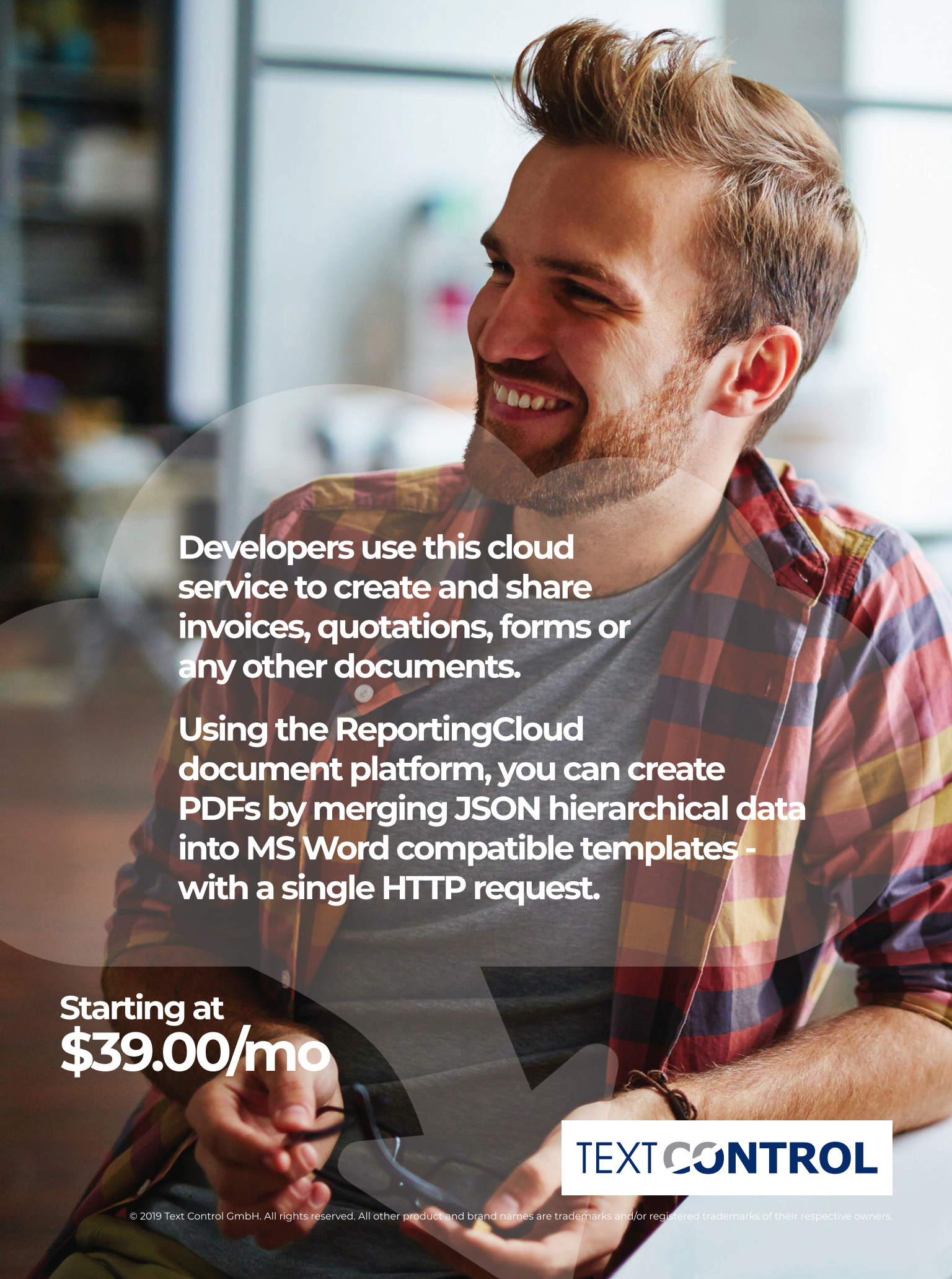
Sponsor a Redmond Intelligence Research Report, a Best Practices guide or a Solution Spotlight report. You'll come away with an authoritative, independent report, professionally edited and designed by the team behind Converge360, which you exclusively distribute. Contact us today for more details.

Redmond INTELLIGENCE
redmondintelligence.com

CONTACT

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

CONVERGE 360
An 1105 MEDIA company



Quantum Messaging with Q# and Blazor

Daniel Vaughan

In this article I explore quantum messaging, leveraging the remarkable phenomenon of quantum entanglement to instantly transfer half a message across potentially vast distances, eliminating the risk of eavesdropping. I'll look at implementing a quantum algorithm for superdense coding in Q#, Microsoft's new quantum programming language; how to entangle qubits using quantum gates; and how to encode ASCII messages as qubits. Then I'll build a Web-based Blazor-powered UI that leverages the quantum algorithm and simulates sending quantum particles to different parties. I'll show you how to consume a Q# library in a Blazor server application and how to launch and coordinate multiple browser windows. You'll also learn how to employ Model-View-ViewModel (MVVM) in a Blazor application.

There's quite a bit of quantum theory used in this article. If you're new to quantum computing, I advise reading my "Quantum Computation Primer." You'll find the first part at tinyurl.com/quantumprimer1.

Let's begin by taking a look at superdense coding.

This article discusses:

- Implementing the superdense coding protocol with Q#
- Implementing MVVM in Blazor with Codon
- Calling Q# APIs with C#
- Launching multiple browser windows in Blazor

Technologies discussed:

Q#, Blazor, JavaScript, C#, Visual Studio 2019 16.2 Preview 1.0

Code download available at:

msdn.com/magazine/0919magcode

Understanding Superdense Coding

Superdense coding leverages the phenomenon of quantum entanglement, where one particle from an entangled pair can affect the shared state of both, despite being separated across potentially vast distances.

To understand the superdense coding protocol, let's say you have three actors: Alice, Bob and Charlie (A, B and C). Charlie creates an entangled pair of qubits and sends one to Alice and one to Bob. When Alice wants to send a 2-bit message to Bob, all she has to do is manipulate her own qubit—which influences the quantum state of the pair—and then send her single qubit to Bob. Bob then measures both Alice's qubit and his own to receive the 2-bit message.

The important point is that there's no way to encode more than one bit of information into a single qubit—yet only one qubit changes hands to deliver a 2-bit message. Despite any distance between the qubits, the shared state of the entangled qubits allows the entire 2-bit message to be encoded using just one of the qubits.

Moreover, when Alice sends Bob a message, a long time may have passed since Charlie sent them each an entangled qubit. You can think of entangled qubits in this way as a resource, waiting to be consumed as part of the communication process.

The pre-sharing of qubits also brings an important security benefit: Anyone who wants to decode the message needs to be in possession of both Alice's and Bob's qubits. Intercepting just Alice's qubit isn't enough to decode the message, and as the entangled qubits are pre-shared, the risk of eavesdropping is eliminated.

If you're skeptical about the physics underpinning superdense coding—and that's a not a bad thing—the phenomenon has been experimentally validated (tinyurl.com/75entanglement), even with satellites and lasers (tinyurl.com/spookyrecord).

Getting Started with Q#

Q# language support for Visual Studio comes with the Microsoft Quantum Development Kit. The easiest way to install the kit for Visual Studio is to use the Visual Studio Extension Manager.

From the Extensions menu in Visual Studio 2019, select Manage Extensions to display the Manage Extensions dialog. Enter "quantum" into the search box to locate the kit. Once it's installed, you can create new Q# project types, Q# syntax highlighting is enabled and debugging of Q# projects works as you'd expect.

Q# is a procedural, domain-specific language, syntactically influenced by C# and F#. Q# files contain operations, functions and custom type definitions, the names of which must be unique within a namespace, as shown in **Figure 1**. Members of other namespaces can be made available using the "open" directive.

Functions are analogous to functions or methods in other procedural languages. A function accepts zero or more arguments and may return a single object, value or tuple. Each parameter to a function consists of a variable name followed by a colon and then its type. The return value for a function is specified after a colon, at the end of the parameter list.

Recall from Part 3 of my "Quantum Computation Primer" (tinyurl.com/controlledgates) that controlled-U gates allow you to add a control input for an arbitrary quantum gate. Q# has a Controlled keyword for this purpose, which allows you to apply a control to an operator, for example:

```
Controlled X([controlQubit1, controlQubit2], targetQubit)
```

Here, the Controlled statement places two control inputs on the Pauli-X gate.

Operations share the same traits as functions, but operations can also represent unitary operators. You can think of them as composite quantum gates. They allow you to define what happens when they're used as part of a controlled operation.

In addition, operations allow you to explicitly define the reverse of the operation (via the adjoint keyword). Adjoint is the complex conjugate transpose of the operator. For more information, see tinyurl.com/brafromket.

The newtype keyword allows you to define a custom type, which you can then use in operations and functions.

Let's now examine some of the most common expression types.

Figure 1 **Functions, Operations and Custom Type Definitions**

```
namespace Quantum.SuperdenseCoding
{
  open Microsoft.Quantum.Diagnostics;
  open Microsoft.Quantum.Intrinsic;
  open Microsoft.Quantum.Canon;

  function SayHellow(name : String) : String
  {
    return "Hello " + name;
  }

  operation EntangledPair(qubit1 : Qubit, qubit2 : Qubit) : Unit
    is Ctl + Adj
  {
    H(qubit1);
    CNOT(qubit1, qubit2);
  }

  newtype IntTuple = (Int, Int);
}
```

Variables are immutable in Q# by default. To declare a new variable, use the let keyword, like so:

```
let foo = 3;
```

The compiler infers the type of the variable from the value you assign it. If you need to change a variable's value, use the mutable keyword when declaring it:

```
mutable foo = 0;
set foo += 1;
```

To create an immutable array in Q#, use the following:

```
let immutableArray = [11, 21, 3, 0];
```

An array in Q# is a value type and is, for the most part, immutable, even when created using the mutable keyword. When the mutable keyword is used, you can assign another array to the identifier, but can't replace items in the array:

```
mutable foo = new Int[4];
set foo = foo w/ 0 <- 1;
```

The w/ operator is an abbreviation of the word "with." On the second line, all the values in foo are copied to a new array, *with* the first item (index 0) in the array set to 1. It has a O(n) runtime complexity. The previous assignment statement can be expressed more succinctly by combining it with an equal sign, like so:

```
set foo w/= 0 <- 1;
```

When the w/= operator is used, where possible an in-place replacement is performed, reducing the runtime complexity to O(1).

You can create a loop by using a for statement:

```
for (i in 1..10)
{
  Message($"i={i},");
}
```

The built-in Message function writes messages to the console. Q# supports string interpolation. By prepending a "$" to a string literal, you can place expressions in curly brackets.

To loop over a collection, use a for statement, like so:

```
for (qubit in qubits) // Qubits is an array of Qubits
{
  H(qubit); // Apply a Hadamard gate to the Qubit
}
```

This was a whirlwind tour of Q#. For more in-depth coverage, please visit tinyurl.com/qsharp. I also recommend working through the Microsoft Quantum Katas (tinyurl.com/quantumkatas).

Implementing Superdense Coding with Q#

Now let's look at implementing the Quantum superdense coding protocol in Q#. I'm assuming you've read the first and second part of my series or already have an understanding of Dirac notation and quantum computation fundamentals.

In the opening description of superdense coding, recall that Charlie entangles a pair of qubits and sends one to Alice and one to Bob.

To entangle a pair of qubits, which start off in a $|0\rangle$ state, you send the first qubit through a Hadamard gate, which changes its state to:

$$|+\rangle = \frac{|01\rangle + |11\rangle}{\sqrt{2}}$$

The Hadamard gate places the first qubit in a superposition of $|0\rangle$ and $|1\rangle$, sending it to the $|+\rangle$ state. Here's the code:

```
operation CreateEntangledPair(qubit1 : Qubit, qubit2 : Qubit) : Unit
{
  H(qubit1);
  CNOT(qubit1, qubit2);
}
```

Bit1	Bit2	Gates	Bell State
0	0	I	$\|\Phi^+\rangle = \frac{\|0_A\,0_B\rangle + \|1_A\,1_B\rangle}{\sqrt{2}}$
0	1	X	$\|\Psi^+\rangle = \frac{\|1_A\,0_B\rangle + \|0_A\,1_B\rangle}{\sqrt{2}}$
1	0	Z	$\|\Phi^-\rangle = \frac{\|0_A\,0_B\rangle - \|1_A\,1_B\rangle}{\sqrt{2}}$
1	1	X,Z	$\|\Psi^-\rangle = \frac{\|1_A\,0_B\rangle - \|0_A\,1_B\rangle}{\sqrt{2}}$

Figure 2 **Gates Used to Produce Bell States**

The CNOT gate inverts qubit2 if the state of qubit1 is $|1\rangle$, resulting in the $|\Phi^+\rangle$ state:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

The state of the qubits when measured will have a 50 percent chance of being either 00 or 11.

With this superposition created, if you were to measure just one of the qubits, it would cause the other's state to collapse to the same value, regardless of the distance between the qubits. The qubits are said to be entangled.

Once Alice and Bob have each been sent one qubit from an entangled pair, Alice can affect the overall quantum state of the two qubits by simply manipulating her own qubit. A single qubit can encode only 1 bit of information. But when entangled with another qubit, that qubit can be used by itself to encode 2 bits of information into the shared quantum state.

Alice is able to transmit 2 bits of information by passing her qubit through one or more gates. The gates and resulting quantum state of the pair of qubits are shown in **Figure 2**. The subscript on the qubits identifies the owner: A for Alice, B for Bob.

> In quantum mechanics, the only perfectly distinguishable states are those that are orthogonal.

Sending a 2-bit message of 00 requires no change to Alice's qubit ("I" is the identity gate, which does nothing). Sending 01 or 10 requires a single gate: X or Z, respectively; and sending 11 requires the application of both an X and a Z gate.

In quantum mechanics, the only perfectly distinguishable states are those that are orthogonal. For qubits, these are states that are perpendicular on the Bloch sphere. It so happens that the Bell states are orthogonal, so when it comes time for Bob to measure the qubits, he can perform an orthogonal measurement on the two qubits to derive Alice's original 2-bit message.

To encode Alice's qubit, it's sent to the EncodeMessageInQubit Q# operation, as shown in **Figure 3**. Along with Alice's qubit, EncodeMessageInQubit accepts two Bool values that represent the 2 bits of information to be sent to Bob. The first let statement concatenates the pair of bit values into an integer between 002 and 112, with bit1 occupying the high bit. The resulting bit sequence indicates the correct gate or gates to apply.

Q# supports many of the language features present in C# and F#. Conveniently, that includes binary literals.

After Alice encodes her message and sends her qubit to Bob, Bob receives the message and applies a CNOT gate to the qubits, and the Hadamard gate to Alice's qubit:

```
operation DecodeMessageFromQubits (bobQubit : Qubit, aliceQubit : Qubit) :
(Bool, Bool)
{
  CNOT(aliceQubit, bobQubit);
  H(aliceQubit);

  return (M(aliceQubit) == One, M(bobQubit) == One);
}
```

Each qubit is then measured. The result is a tuple consisting of the two original message bits sent by Alice. The measured values of each qubit correspond to the classical bit message encoded by Alice.

Let's look more closely at how this superdense coding protocol works. Recall that both Alice and Bob are each given a qubit belonging to an entangled pair. The matrix representation of the quantum state of the two qubits is given by:

$$CNOT|(H|0\rangle \otimes |0\rangle) = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\1\end{bmatrix}$$

I'll use this matrix result in a moment to demonstrate the encoding process.

If Alice wishes to send the message 012 to Bob, she applies the X gate to her own qubit, which changes the quantum state to:

$$|\Psi\rangle = \frac{(X|0\rangle \otimes |0\rangle) + (X|1\rangle \otimes |1\rangle)}{\sqrt{2}}$$

$$= \frac{(|1\rangle \otimes |0\rangle) + (|0\rangle \otimes |1\rangle)}{\sqrt{2}}$$

$$= \frac{|10\rangle + |01\rangle}{\sqrt{2}}$$

Figure 3 **EncodeMessageInQubit Operation**

```
operation EncodeMessageInQubit(
  aliceQubit : Qubit, bit1 : Bool, bit2 : Bool) : Unit
{
  let bits = (bit1 ? 1 | 0) * 2 + (bit2 ? 1 | 0);

  if (bits == 0b00)
  {
    I(aliceQubit);
  }
  elif (bits == 0b01)
  {
    X(aliceQubit);
  }
  elif (bits == 0b10)
  {
    Z(aliceQubit);
  }
  elif (bits == 0b11)
  {
    X(aliceQubit);
    Z(aliceQubit);
  }
}
```

Converting that to its matrix form yields:

$$= \frac{1}{\sqrt{2}}\left(\begin{bmatrix}0\\0\\1\\0\end{bmatrix} + \begin{bmatrix}0\\1\\0\\0\end{bmatrix}\right) = \begin{bmatrix}0\\1\\1\\0\end{bmatrix}$$

Alice then sends her qubit to Bob.

To decode the message, Bob begins by applying a CNOT gate to the qubits:

$$CNOT|\Psi\rangle = CNOT\left(\frac{|10\rangle + |01\rangle}{\sqrt{2}}\right)$$

You can now take the matrix result from the previous step and multiply it by the CNOT matrix, remembering that the scalar multiplier is commutable:

$$= \frac{1}{\sqrt{2}}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}0\\1\\1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\1\\0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\left(\begin{bmatrix}0\\1\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\1\end{bmatrix}\right)$$

When you factor the resulting matrix, you get:

$$= \frac{|01\rangle + |11\rangle}{\sqrt{2}}$$

Bob then applies a Hadamard gate to Alice's qubit:

$$|\Psi\rangle = \frac{H|0\rangle \otimes |1\rangle + H|1\rangle \otimes |1\rangle}{\sqrt{2}}$$

You can substitute in the matrix representation of the Hadamard gate and qubits:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}0\\0\end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}$$

Dividing top and bottom by $\sqrt{2}$, you eliminate the common divisor, which leaves some matrix arithmetic to do:

$$= \begin{bmatrix}1&1\\1&-1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}1&1\\1&-1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}$$

$$= \begin{bmatrix}1\\1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}1\\-1\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}$$

$$= \begin{bmatrix}0\\1\\0\\1\end{bmatrix} + \begin{bmatrix}0\\1\\0\\-1\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = |01\rangle$$

As you can see, the final result is the original message sent by Alice!

When Alice passed her qubit through the X gate, it affected the overall superposition of the two qubits. Bob was able to determine the change Alice made by unwinding the superposed state.

Allocating Qubits

When a Qubit object is allocated in Q#, it must be returned to the simulator. This is enforced syntactically. A using statement is required to allocate qubits.

To allocate a single Qubit, use the Qubit expression, as shown here:

```
using (qubit = Qubit())
{
  Reset(qubit);
}
```

The Reset function returns the Qubit to the $|0\rangle$ state.

When exiting the using block, a qubit must be in the $|0\rangle$ state. This is a runtime requirement, enforced by the quantum simulator. You can pass a Qubit to other operations and change its state and so forth, but if execution reaches the end of the using block without the Qubit being returned to the $|0\rangle$ state, an exception is raised. You can disable exceptions for non-reset qubits via the QuantumSimulator constructor.

You can also create multiple Qubit objects using the array expression Qubit[]:

```
using (qubits = Qubit[2])
{
  // Do something with the qubits...
  ResetAll(qubits);
}
```

The qubits created with the array expression still need to be reset when you're done. Use the built-in ResetAll function to reset multiple qubits.

Be aware that simulated qubits use an exponential amount of memory. Each allocated qubit doubles the amount of RAM required. In my tests on a machine with 16GB of RAM, usage got to around 13GB for 26 qubits. Exceeding that, virtual memory kicks in. I was able to achieve 31 qubits; however, the memory usage ballooned to 53GB. Unsurprisingly, 32 qubits produced an interop exception.

Bending Q# to Your Will

The example project passes qubits around asynchronously among objects that represent Alice, Bob and Charlie. For that reason, I needed to find a way to retain Qubit objects and to circumvent Q#'s requirement that a using block be used.

There's a strong affinity between the Q# and C# tooling in Visual Studio. When you build a Q# project, the .qs files in that project are translated into C#. The Q# tooling outputs its generated C# files with a ".g.cs" file extension, placing them in the \obj\qsharp\src directory of the parent project.

> When a Qubit object is allocated in Q#, it must be returned to the simulator. This is enforced syntactically.

I suspect this translation step might be replaced with direct compilation to IL or something else in the future, given the investment Microsoft is making in the Q# compiler. For the moment, however, I found it useful to explore the generated .cs files by peeking at how things work behind the scenes. I soon realized that it's possible to leverage the Q# SDK assemblies from C# to achieve what I needed.

The QOperations class, in the downloadable sample code, directly calls the Q# APIs, which allows it to request a Qubit from the simulator without having to immediately release it. Instead

Figure 4 **Creating Entangled Qubits in C#**

```
public Task<IList<(Qubit, Qubit)>> GetEntangledPairsAsync(int count)
{
  IList<(Qubit, Qubit)> result = new List<(Qubit, Qubit)>(count);

  for (int i = 0; i < count; i++)
  {
    var qubits = allocator.Apply(2);
    hGate.Apply(qubits[0]);
    cnotGate.Apply((qubits[0], qubits[1]));
    result.Add((qubits[0], qubits[1]));
  }

  return Task.FromResult(result);
}
```

Figure 5 **Releasing Qubits in C#**

```
public Task ReleaseQubitsAsync(IEnumerable<Qubit> qubits)
{
  foreach (Qubit qubit in qubits)
  {
    resetOperation.Apply(qubit);
    releaseOperation.Apply(qubit);
    /* Alternatively, we could do: */
    // simulator.QubitManager.Release(qubit);
  }

  return Task.CompletedTask;
}
```

of restricting myself to Q# for all quantum activities, I use C# for allocating, entangling and deallocating qubits.

The QOperations class creates an instance of the QuantumSimulator class, which is located in the Microsoft.Quantum.Simulation.Simulators namespace. QuantumSimulator provides an API for manipulating the native simulator component. The simulator is installed with the Microsoft Quantum Development Kit.

QuantumSimulator has an internal IoC container that allows you to retrieve various objects that are commonly used by its generated C# code. For example, within a C# file you can use the Get<T> method of the QuantumSimulator object to retrieve an object for resetting a qubit's state and releasing a qubit, performing the same action as a using block in Q#:

```
resetOperation = simulator.Get<ICallable<Qubit, QVoid>>(typeof(Reset));
releaseOperation = simulator.Get<Release>(typeof(Release));
```

To allocate a new Qubit, you use the Microsoft.Quantum.Intrinsic.Allocate object. You also need the CNOT and H (Hadamard) gate objects, which are retrieved from the container, like so:

```
allocator = simulator.Get<Allocate>(typeof(Allocate));
cnotGate = simulator.Get<IUnitary<(Qubit, Qubit)>>(typeof(CNOT));
hGate = simulator.Get<IUnitary<Qubit>>(typeof(H));
```

With these objects you can generate entangled pairs of qubits. The GetEntangledPairsAsync method of the QOperations class creates a list of Qubit tuples (see **Figure 4**).

I use the Apply method of the Allocate object to retrieve two free qubits from the simulator. I apply the Hadamard gate to the first qubit, and then the CNOT to both, placing them in an entangled state. There's no need to reset and release them immediately; I'm able to return the Qubit pairs from the method. Qubits still need to be released—otherwise the application would quickly run out of memory. I just postpone that step.

Qubits are released with the ReleaseQubitsAsync method, as shown in **Figure 5**.

For each supplied Qubit, I reset the qubit using the Reset object's Apply method. I then inform the simulator that the qubit is free using the Release object's Apply method. Alternatively, you could use the simulator's QubitManager property to release the qubit.

That completes the quantum element of this article. Now let's look at Blazor and at implementing a server-side Blazor project with MVVM.

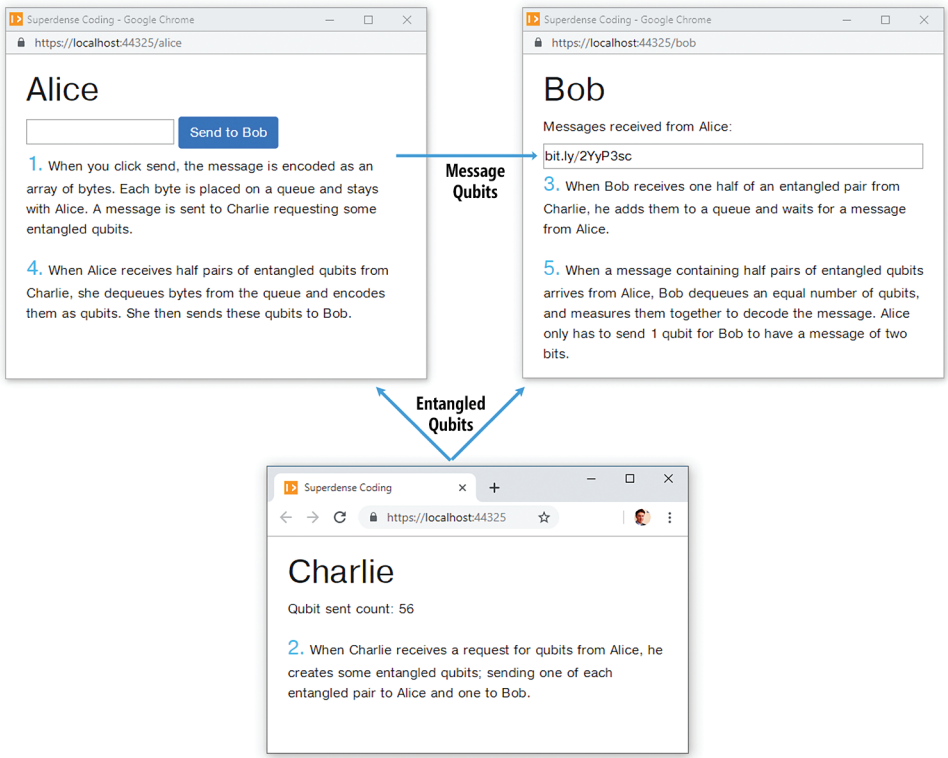


Figure 6 **Alice, Bob and Charlie Razor Pages**

Using the MVVM Pattern with Blazor

I've been a longtime fan of XAML. I like the way I can write view-model logic in C# and combine it with a designer-friendly reusable layout file glued together with data bindings. Blazor allows the same approach, but instead of XAML it uses Razor, which has a concise syntax for marking up dynamic content within HTML.

If you're new to Blazor, see the installation instructions at tinyurl.com/installblazor for your IDE of choice.

I first implemented this project using Universal Windows Platform (UWP). UWP is compatible with .NET Core and it made sense to rapidly build out the UI in a XAML-based technology. In fact, the UWP version of the application is located in the

Figure 7 **Charlie.razor Function Block Excerpt**

```
@functions {
  bool scriptInvoked;
  ...
  protected override void OnAfterRender()
  {
    if (!scriptInvoked)
    {
      scriptInvoked = true;
      jsRuntime.InvokeAsync<object>("eval", evalScript);
    }
  }

  const string evalScript = @"var w1 = window.open('/alice', '_blank',
    'left=100,top=50,width=500,height=350,toolbar=0,resizable=0');
var w2 = window.open('/bob', '_blank',
  'left=100,top=500,width=500,height=350,toolbar=0,resizable=0');
window.addEventListener('beforeunload', function (e) {
  try {
    w1.close();
  } catch (err) {}
  try {
    w2.close();
  } catch (err) {}
  (e || window.event).returnValue = null;
  return null;
});";
}
```

downloadable sample code. If you don't feel like installing Blazor, and just wish to see the app in action, feel free to just run the UWP version. Both applications are identical in behavior.

Porting the app to Blazor was a piece of cake. No code changes were required; NuGet references to the Codon (codonfx.com) MVVM framework were fine and everything just worked.

I chose Blazor's server-side model for this project to allow me to share a single QuantumSimulator object among the three view-models representing Alice, Bob, and Charlie, and to pass Qubit objects around.

Dissecting the Blazor Application The Blazor application contains three Razor pages, shown in **Figure 6**, representing the three actors: Alice.razor, Bob.razor and Charlie.razor.

Each page has an associated view-model. All three view-models in the project subclass Codon's ViewModelBase class, which provides built-in support for INotifyPropertyChanged (INPC) events, an IoC container reference, and loosely coupled messaging between components within the application.

Each razor page has a functions block that includes a property for its respective view-model. Here's the functions block in Alice.razor:

```
@functions {
  AliceViewModel ViewModel { get; set; }

  protected override async Task OnInitAsync()
  {
    ViewModel = Dependency.Resolve<AliceViewModel>();
    ViewModel.PropertyChanged += (o, e) => Invoke(StateHasChanged);
  }
}
```

The view-model is retrieved from Codon's IoC container during the OnInitAsync method.

View-model property changes don't automatically cause the page content to update. For that you subscribe to the ViewModelBase classes PropertyChanged event. When the event occurs, the razor page's StateHasChanged method is called, which triggers an update of the HTML elements that are data-bound to the view-model.

You use the page's Invoke method to ensure that updates occur on the UI thread. If the StateHasChanged method is called from a non-UI thread, an exception is raised.

Requesting Qubits from Charlie The Charlie razor page indicates the number of qubits sent to Alice and Bob. As **Figure 7** shows, this page has slightly more going on in its functions block.

Besides view-model initialization, the page is also tasked with opening a new browser window for both Alice and Bob. It does this by calling JSRuntime.InvokeAsync and using the JavaScript eval function to run JavaScript in the browser. The eval script opens and retains the windows as variables. When the Charlie page's beforeunload event occurs, the windows are closed. This prevents the accumulation of orphaned browser windows when you stop the debugger.

Be aware that most browsers today block popups by default, or at least request consent before opening one. If this happens, you'll need to OK the popups and refresh the page. My use of popups in this project is merely as a convenience to open all three actor windows at once.

The Charlie.razor page contains a single dynamic element—a counter that displays the number of qubits that have been dispatched:

```
<p>Qubit sent count: @Model.QubitSentCount</p>
```

The value within the paragraph is bound to the view-model's QubitSentCount property.

When classes derive from Codon.UIModel.ViewModelBase, any implementation of IMessageSubscriber<T> automatically subscribes to messages of type T. CharlieViewModel implements IMessageSubscriber<RequestQubitsMessage> and IMessageSubscriber<ReleaseQubitsMessage>; therefore, when a RequestQubitsMessage is published by the AliceViewModel or a ReleaseQubitsMessage is published by BobViewModel, it's handled within the CharlieViewModel class. This is all Charlie is tasked with; he sends out entangled qubits to Alice and Bob when they're needed.

When CharlieViewModel receives a RequestQubitsMessage, it uses the QOperations instance to retrieve a list of entangled qubit pairs, as shown in **Figure 8**.

CharlieViewModel then sends the first item of each pair to Alice, and the second to Bob, via the IMessenger.

Finally, QubitSentCount is increased, which updates the Charlie page.

Sending Messages as Alice Let's take a look at the Alice page and its associated view-model. The AliceViewModel class contains an AsyncActionCommand, which is defined like so:

```
ICommand sendCommand;

public ICommand SendCommand => sendCommand ??
        (sendCommand = new AsyncActionCommand(SendAsync));
```

Figure 8 **CharlieViewModel ReceiveMessageAsync (RequestQubitsMessage) Method**

```
async Task IMessageSubscriber<RequestQubitsMessage>.ReceiveMessageAsync(
  RequestQubitsMessage message)
{
  IList<(Qubit, Qubit)> qubits =
    await QOperations.GetEntangledPairsAsync(message.QubitCount);

  await Messenger.PublishAsync(new BobQubitMessage(qubits.Select(x => x.Item2)));
  await Messenger.PublishAsync(new AliceQubitMessage(qubits.Select(x => x.Item1)));

  QubitSentCount += message.QubitCount;
}
```

Figure 9 **AliceViewModel.SendAsync Method**

```
async Task SendAsync(object arg)
{
  var bytes = Encoding.ASCII.GetBytes(Message);

  foreach (byte b in bytes)
  {
    byteQueue.Enqueue(b);
    await Messenger.PublishAsync(
      new RequestQubitsMessage(qubitsRequiredForByte));
  }

  Message = string.Empty;
}
```

I'm still chuffed seeing properties written with such conciseness, where a lambda expression combined with a null coalescing operator brings lazy loading in just a couple lines of code.

Codon's command infrastructure supports async command handlers. By using Codon's AsyncActionCommand, you can specify the async method SendAsync as the command handler.

Alice.razor contains a text field and a button:

```
<input type="text" bind="@ViewModel.Message" />
<button class="btn btn-primary"
      onclick="@ViewModel.SendCommand.Execute">Send to Bob</button>
```

The input box is bound to the view-model's Message property. When the *Send to Bob* button is clicked, the view-model's SendCommand executes, and its SendAsync method is called (see **Figure 9**).

The view-model has a queue that contains the bytes of each message that's encoded and sent to Bob. Codon's IMessenger is used to send off a message, which ultimately reaches Charlie, requesting four qubits be entangled and sent out, two for Alice and two for Bob.

Finally, the Message property is reset to string.empty, which clears the input box and leaves it ready for a new message.

AliceViewModel implements IMessageSubscriber<AliceQubitMessage> and thus receives all messages of that type:

```
gasync Task IMessageSubscriber<AliceQubitMessage>.ReceiveMessageAsync(
  AliceQubitMessage message)
{
  foreach (var qubit in message.Qubits)
  {
    qubitQueue.Enqueue(qubit);
  }

  await DispatchItemsInQueue();
}
```

When ReceiveMessageAsync is called, the payload contains a collection of Qubit instances from Charlie. AliceViewModel also retains a queue of Qubit objects, to which the newly arrived qubits are added.

AliceViewModel is now ready to send at least part of a message to Bob. DispatchItemsInQueue is called, as shown in **Figure 10**.

DispatchItemsInQueue first retrieves the QOperations instance from the IoC container. Dependency.Resolve<T,T>(bool singleton) causes a new instance to be created if one hasn't been already. That instance is then retained as a singleton so that future requests will resolve the same object.

An extension method is used to dequeue four qubits from the qubit queue. The message byte is also dequeued and placed into a BitArray. The QOperations object is then tasked with placing each qubit into a superposition representing the 2-bit message. Notice again how one qubit is able to represent two classical bits because its state affects the quantum state of the pair.

Figure 10 **AliceViewModel.DispatchItemsInQueue Method**

```
async Task DispatchItemsInQueue()
{
  var qOps = Dependency.Resolve<QOperations, QOperations>(true);

  while (byteQueue.Any())
  {
    if (qubitQueue.Count < qubitsRequiredForByte)
    {
      return;
    }

    IList<Qubit> qubits =
      qubitQueue.DequeueMany(qubitsRequiredForByte).ToList();

    byte b = byteQueue.Dequeue();
    BitArray bitArray = new BitArray(new[] { b });

    /* Convert classical bit pairs to single qubits. */
    for (int i = 0, j = 0; i < bitArray.Length; i += 2, j++)
    {
      await qOps.EncodeMessageInQubitAsync(
        qubits[j],
        bitArray[i],
        bitArray[i + 1]);
    }

    await Messenger.PublishAsync(new DecodeQubitsMessage(qubits));
  }
}
```

Figure 11 **ReceiveMessageAsync (DecodeQubitsMessage) Method**

```
async Task IMessageSubscriber<DecodeQubitsMessage>.ReceiveMessageAsync(
  DecodeQubitsMessage message)
{
  IList<Qubit> aliceQubits = message.Qubits;
  List<Qubit> bobQubits = qubits.DequeueMany(aliceQubits.Count).ToList();

  var qOps = Dependency.Resolve<QOperations, QOperations>(true);
  var bytes = new List<byte>();

  for (int i = 0; i < bobQubits.Count; i += 4)
  {
    byte b = 0;

    for (int j = 0; j < 4; j++)
    {
      (bool aliceBit, bool bobBit) = await qOps.DecodeQubits(
        bobQubits[i + j], aliceQubits[i + j]);

      if (bobBit)
      {
        b |= (byte)(1 << (j * 2 + 1));
      }

      if (aliceBit)
      {
        b |= (byte)(1 << (j * 2));
      }
    }

    bytes.Add(b);
  }

  Message += Encoding.ASCII.GetString(bytes.ToArray());

  await Messenger.PublishAsync(new ReleaseQubitsMessage(aliceQubits));
  await Messenger.PublishAsync(new ReleaseQubitsMessage(bobQubits));
}
```

The IMessenger is then used to dispatch a DecodeQubitsMessage that's ultimately received by Bob.

Receiving Messages as Bob Like Alice, Bob has a queue of Qubits that arrive from Charlie. BobViewModel implements IMessageSubscriber<BobQubitMessage>. When Charlie sends

References

- "Statements and Other Constructs," retrieved on July 5, 2019, from bit.ly/20fx1Ld
- Yanofsky, N., & Mannucci, M. (2008) "Quantum Computing for Computer Scientists," Cambridge University Press
- Nielsen, M. & Chuang, I., (2010) "Quantum Computation and Quantum Information," 10th edition, Cambridge University Press, Cambridge UK
- Watrous, J., (2006) "Lecture 3: Superdense coding, quantum circuits, and partial measurements," retrieved on July 5, 2019, from bit.ly/2XTPEDN

qubits wrapped in a BobQubitMessage, they're placed in the queue.

BobViewModel also implements IMessageSubscriber<DecodeQubitsMessage>. When Alice sends a message to Bob, it comes wrapped in a DecodeQubitsMessage object and is handled in the ReceiveMessageAsync(DecodeQubitsMessage) method (see **Figure 11**). The method dequeues an equal number of qubits from its queue. The QOperations DecodeQubits method is used to convert each pair of qubits from Alice and Bob into 2 bits of data. Each byte in the message consists of four 2-bit pairs (8 bits), hence the loop increments for i and j.

Each byte is constructed by left-shifting bits according to the values of the decoded bits. The decoded bytes are turned back into a string using the Encoding.ASCII.GetString method and appended to the Message property that's displayed on the page.

Once the qubits have been decoded, they're released by publishing a ReleaseQubitsMessage, which is received by the CharlieViewModel. QOperations is then used to release the qubits:

```
async Task IMessageSubscriber<ReleaseQubitsMessage>.ReceiveMessageAsync(
  ReleaseQubitsMessage message)
{
  await QOperations.ReleaseQubitsAsync(message.Qubits);
}
```

Wrapping Up

In this article I implemented a quantum algorithm for superdense coding in Q#, the new Microsoft quantum programming language. I explained how to entangle qubits using quantum gates, and how to encode ASCII messages as qubits. I looked at a Web-based Blazor-powered UI that leveraged the quantum algorithm and simulated sending quantum particles to different parties. I also showed you how to consume a Q# library in a Blazor server application and how to launch and coordinate multiple browser windows. Finally, I explained how to employ MVVM in a Blazor application. ■

Daniel Vaughan *is an author and software engineer working for Microsoft in Redmond, Wash. He's a former nine-time Microsoft MVP, and is the developer behind several acclaimed consumer and enterprise mobile apps, such as Surfy Browser for Android and Windows Phone and Airlock Browser for Android. He's also the creator of a number of popular open source projects, including the Codon and Calcium frameworks. Vaughan blogs at danielvaughan.org and tweets as @dbvaughan.*

Thanks to the following Microsoft technical expert for reviewing this article: *Bettina Heim (Microsoft Quantum Team)*

AGENDA-AT-A-GLANCE

#LIVE360

Business Intelligence	SQL Server Administration & Maintenance	SQL Server Features & Components	SQL Server for Developers	SQL Server Performance Tuning and Optimization

START TIME	END TIME			
SQL Server Live! Full Day Hands-On Lab: Sunday, November 17, 2019				
7:15 AM	9:00 AM	Registration • Coffee and Morning Pastries		
9:00 AM	6:00 PM	**SQS01** Hands-On Lab: The A to Z of Cloud-based SQL Server Solutions - *Allan Hirt*		
2:00 PM	7:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center		
START TIME	**END TIME**	**SQL Server Live! Pre-Conference Workshops: Monday, November 18, 2019**		
7:00 AM	8:30 AM	Registration • Coffee and Morning Pastries		
8:30 AM	5:30 PM	**SQM01** Workshop: Advanced Data Protection - Security and Privacy in SQL Server - *Thomas LaRock & Karen Lopez*	**SQM02** Workshop: SQL Server 2019 - *Leonard Lobel*	
6:30 PM	8:00 PM	Dine-A-Round Dinner @ Universal CityWalk - 6:30pm - Meet at Conference Registration Desk to walk over with the group		
START TIME	**END TIME**	**SQL Server Live! Day 1: Tuesday, November 19, 2019**		
7:00 AM	8:00 AM	Registration • Coffee and Morning Pastries		
8:00 AM	9:00 AM	**SQL SERVER LIVE! KEYNOTE: To Be Announced**		
9:15 AM	10:30 AM	**SQT01** AWS Versus Azure: Data Services Comparison - *Thomas LaRock*	**SQT02** Availability Fundamentals for SQL Server - *Allan Hirt*	**SQT03** Power BI: What Have You Done For Me Lately - *Andrew Brust*
10:30 AM	11:00 AM	Networking Break • Visit the EXPO - *Pacifica 7*		
11:00 AM	12:00 PM	**LIVE! 360 KEYNOTE: To Be Announced** - *Jessica Deen, Azure Avenger, Microsoft & Abel Wang, Senior Cloud Developer Advocate, Microsoft*		
12:00 PM	12:45 PM	Lunch • Visit the EXPO		
12:45 PM	1:30 PM	Dessert Break • Visit the EXPO		
1:30 PM	1:50 PM	**SQT04** Fast Focus: Automation for the DBA: Embrace Your Inner Sloth - *William Durkin*	**SQT05** Fast Focus: Who's Tinkling in Your Data Lake? - *Karen Lopez*	
2:00 PM	2:20 PM	**SQT06** Fast Focus: From Adaptive to Intelligent: Query Processing in SQL 2019 - *Hugo Kornelis*	**SQT07** Fast Focus: Performance Tuning Without Changing Code - *Thomas LaRock*	
2:20 PM	2:45 PM	Networking Break • Visit the EXPO - *Pacifica 7*		
2:45 PM	4:00 PM	**SQT08** SQL Server 2019 Deep Dive - *Scott Klein*	**SQT09** SQL Server In-Memory Database Objects - *Denny Cherry*	**SQT10** What's New in SQL Server 2019 for Analytics - *Thomas LeBlanc*
4:15 PM	5:30 PM	**SQT11** Microsoft's New Database Experimentation Assistant (DEA) - *Mindy Curnutt*	**SQT12** Common Troubleshooting Techniques for Availability Groups and Failover Cluster Instances - *Allan Hirt*	**SQT13** Power BI implementation guidance: from Pro to Premium and Desktop to Embedded - *Thomas LeBlan*
5:30 PM	7:30 PM	Exhibitor Reception - *Pacifica 7*		
START TIME	**END TIME**	**SQL Server Live! Day 2: Wednesday, November 20, 2019**		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries		
8:00 AM	9:15 AM	**SQW01** An Introduction to Spatial Data in SQL Server - *Mindy Curnutt*	**SQW02** It's Broken, Now What?! Practical Problem Solving - *William Durkin*	**SQW03** Getting Started with Apache Spark - *Kevin Feasel*
9:30 AM	10:45 AM	**SQW04** Blockchain for the DBA & Data Professional - *Karen Lopez*	**SQW05** SQL Server Open Query Store - *William Durkin*	**SQW06** PolyBase in Action - *Kevin Feasel*
10:45 AM	11:30 AM	Networking Break • Visit the EXPO - *Pacifica 7*		
11:30 AM	12:30 PM	**LIVE! 360 KEYNOTE: To Be Announced**		
12:30 PM	1:30 PM	Birds-of-a-Feather Lunch		
1:30 PM	2:00 PM	Dessert Break • Visit the EXPO		
2:00 PM	3:15 PM	**SQW07** SQL Data Discovery and Classification - *Karen Lopez*	**SQW08** Things You Should Never Do In Microsoft SQL Server - *Denny Cherry*	**SQW09** Deep Dive into Power BI Administration and Security Internals - *Ginger Grant*
3:15 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m. - *Pacifica 7*		
4:00 PM	5:15 PM	**SQW10** Migrating SSIS Workloads to Azure Data Factory - *Joshua Luedeman*	**SQW11** Improve Your Database Performance in Seven Simple Steps - *Hugo Kornelis*	**SQW12** Performance Tuning Power BI - Ginger Grant
7:30 PM	9:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*		
START TIME	**END TIME**	**SQL Server Live! Day 3: Thursday, November 21, 2019**		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries		
8:00 AM	9:15 AM	**SQH01** Deep Dive into Adaptive Query Processing - *Hugo Kornelis*	**SQH02** Using Modular Scripts to Perform SQL Compliance Audits in Seconds - *Chris Bell*	**SQH03** DataOps and Business Intelligence Testing Approaches - *Jen Stirrup*
9:30 AM	10:45 AM	**SQH04** Data Security and Privacy Techniques for Modern Databases - *Thomas LaRock*	**SQH05** Building Your First SQL Server Container Lab in Docker - *Chris Bell*	**SQH06** To Be Announced
11:00 AM	12:00 PM	**SQL SERVER LIVE! PANEL DISCUSSION: SQL Server is Dead, Long Live SQL Server!** *Thomas LaRock (moderator), Karen Lopez, Bradley Ball, Jen Stirrup, Joseph D'Antoni*		
12:00 PM	1:00 PM	Lunch - *Oceana Ballroom*		
1:00 PM	2:15 PM	**SQH07** Machine Learning with R in Azure SQL Database - *Bradley Ball*	**SQH08** Building a Better Data Solution: Microsoft SQL Server and Azure Data Services - *Joseph D'Antoni*	**SQH09** Data Preparation: A framework with Power BI, Power Query, and Power BI Dataflows - *Jen Stirrup*
2:30 PM	3:45 PM	**SQH10** TimescaleDB on Azure Database for PostgreSQL - *Bradley Ball*	**SQH11** Containers, Pods, and Databases - Learning About the Future of Infrastructure - *Joseph D'Antoni*	**SQH12** Getting Started with Power BI Report Server - *Joshua Luedeman*
4:00 PM	5:00 PM	Next? Live! 360 Networking Event *Thomas LaRock, Bradley Ball, Joseph D'Antoni, Karen Lopez & Jen Stirrup*		
START TIME	**END TIME**	**SQL Server Live! Post-Conference Workshops: Friday, November 22, 2019**		
7:30 AM	8:00 AM	Registration • Coffee and Morning Pastries		
8:00 AM	5:00 PM	**SQF01** Workshop: Azure SQL Data Warehouse: A Full Day of the Transformative New NDA Features - *Bradley Ball, Joshua Luedeman, & Gareth Swanepoel*	**SQF02** Workshop: SQL Server High Performance Development - *Joseph D'Antoni*	

Speakers and sessions subject to change

PRODUCED BY
CONVERGE360 an 1105 MEDIA company

CONNECT WITH LIVE! 360
twitter.com/live360 @live360
facebook.com Search "Live 360"
instagram.com @live360_events
linkedin.com Join the "Live! 360" group!

Expression Trees in Visual Basic and C#

Zev Spitz

Imagine if you had objects that represented parts of a program. Your code could combine these parts in various ways, creating an object that represented a new program. You could compile this object into a new program and run it; your program could even rewrite its own logic. You could pass this object to a different environment, where the individual parts would be parsed as a set of instructions to be run in this other environment.

Welcome to expression trees. This article aims to provide a high-level overview of expression trees, their use in modeling code operations and runtime code compilation, and how to create and transform them.

A Note About Language For many language features in Visual Basic and C#, the syntax is actually just a thin wrapper over language-agnostic .NET types and methods. For example, C#'s foreach loop and Visual Basic's For Each construct both call into the IEnumerable-implementing type's GetEnumerator method. LINQ keyword syntax is resolved into methods such as Select and Where with the appropriate signature. A call to IDisposable.Dispose wrapped in error handling is generated every time you write Using in Visual Basic or a using block in C#.

This is true of expression trees, as well—both Visual Basic and C# have syntactic support for creating expression trees, and can use the .NET infrastructure (in the System.Linq.Expressions namespace) for working with them. As such, the concepts described here apply equally well to both languages. Whether your primary development language is C# or Visual Basic, I believe you'll find value in this article.

This article discusses:

- Mapping code to APIs with expression trees
- Constructing expression trees
- Expression tree visitors
- Runtime code compilation

Technologies discussed:

Expression Trees, C#, Visual Basic .NET

Code download available at:

bit.ly/2yku7tx

Expressions, Expression Trees and the Types in System.Linq.Expressions

An expression in Visual Basic and C# is a piece of code that returns a value when evaluated, for example:

```
42
"abcd"
True
n
```

Expressions can be composed of other expressions, such as:

```
x + y
"abcd".Length < 5 * 2
```

These form a tree of expressions, or an *expression tree*. Consider n + 42 = 27 in Visual Basic, or n + 42 == 27 in C#, which can be separated into parts as shown in **Figure 1**.

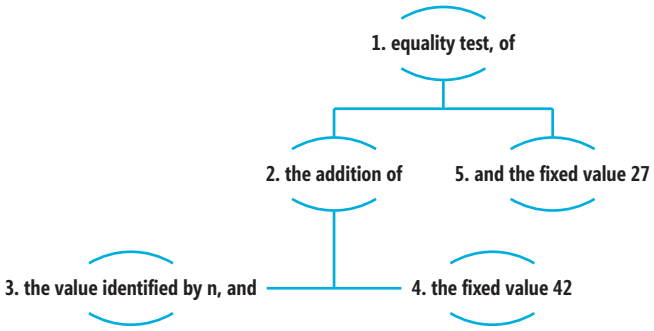


Figure 1 **Breakdown of an Expression Tree**

.NET provides a set of types in the System.Linq.Expressions namespace for constructing data structures that represent expression trees. For example, n + 42 = 27 can be represented with objects and properties as shown in **Figure 2**. (Note that this code will not compile—these types don't have public constructors, and they're immutable, so no object initializers.)

The term expression tree in .NET is used for both syntactical expression trees (x + y), and expression tree objects (a BinaryExpression instance).

To understand what a given node in the expression tree represents, the node object's type—BinaryExpression, ConstantExpression, ParameterExpression—only describes the *shape* of the expression. For example, the BinaryExpression type tells me that the represented expression has two operands, but not which operation combines the operands. Are the operands added together, multiplied together or compared numerically? That information is contained in the NodeType property, defined in the Expression base class. (Note that some node types don't represent code operations—see "Meta Node Types.")

An additional property of note on Expression is the Type property. In both Visual Basic and C#, every expression has a type—adding two Integers will produce an Integer, while adding two Doubles will produce a Double. The Type property returns the System.Type of the expression.

When visualizing the structure of an expression tree, it's useful to focus on these two properties, as depicted in **Figure 3**.

If there are no public constructors for these types, how do we create them? You have two choices. The compiler can generate them for you if you write lambda syntax where an expression type is expected. Or you can use the Shared (static in C#) factory methods on the Expression class.

Constructing Expression Tree Objects I: Using the Compiler

The simple way to construct these objects is to have the compiler generate them for you, as mentioned previously, by using lambda syntax anywhere an Expression(Of TDelegate) (Expression<TDelegate> in C#) is expected: either assignment to a typed variable, or as an argument to a typed parameter. Lambda syntax is introduced in Visual Basic using either the Function or Sub keywords, followed by a parameter list and a method body; in C# the => operator indicates lambda syntax.

Figure 2 **Expression Tree Objects Using Object Notation**

```
' Visual Basic
New BinaryExpression With {
  .NodeType = ExpressionType.Equal,
  .Left = New BinaryExpression With {
    .NodeType = ExpressionType.Add,
    .Left = New ParameterExpression With {
      .Name = "n"
    },
    .Right = New ConstantExpression With {
      .Value = 42
    }
  },
  .Right = New ConstantExpression With {
    .Value = 27
  }
}

// C#
new BinaryExpression {
  NodeType = ExpressionType.Equal,
  Left = new BinaryExpression {
    NodeType = ExpressionType.Add,
    Left = new ParameterExpression {
      Name = "n"
    },
    Right = new ConstantExpression {
      Value = 42
    }
  },
  Right = new ConstantExpression {
    Value = 27
  }
}
```

Lambda syntax that defines an expression tree is called an expression lambda (as opposed to a statement lambda), and looks like this in C#:

```
Expression<Func<Integer, String>> expr = i => i.ToString();

IQueryable<Person> personSource = ...
var qry = qry.Select(x => x.LastName);
```

And like this in Visual Basic:

```
Dim expr As Expression(Of Func(Of Integer, String))  = Function(i) i.ToString

Dim personSource As IQueryable(Of Person) = ...
Dim qry = qry.Select(Function(x) x.LastName)
```

Producing an expression tree in this way has some limitations:

- Limited by lambda syntax
- Supports only single-line lambda expressions, with no multiline lambdas
- Expression lambdas can only contain expressions, not statements such as If…Then or Try…Catch
- No late binding (or dynamic in C#)
- In C#, no named arguments or omitted optional arguments
- No null-propagation operator

The complete structure of the constructed expression tree—that is, node types, types of sub-expressions and overload resolution of methods—is determined at compile time. Because expression trees are immutable, the structure cannot be modified once created.

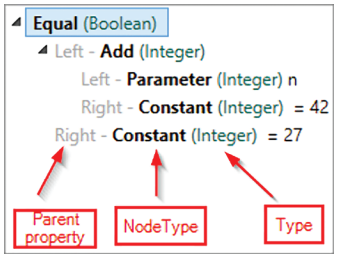


Figure 3 **NodeType, Type, Value and Name Properties**

Note also that compiler-generated trees mimic the

behavior of the compiler, so the generated tree might be different from what you'd expect given the original code (**Figure 4**). Some examples:

Closed-over Variables In order to reference the current value of variables when going in and out of lambda expressions, the compiler creates a hidden class whose members correspond to each of the referenced variables; the lambda expression's function becomes a method of the class (see "Closed-over Variables"). The corresponding expression tree will render closed-over variables as MemberAccess nodes on the hidden instance. In Visual Basic the VBLocal_ prefix will also be appended to the variable name.

NameOf Operator The result of the NameOf operator is rendered as a Constant string value.

String Interpolation and Boxing Conversion This is resolved into a call to String.Format and a Constant format string. Because the interpolation expression is typed as a value type—Date (the Visual Basic alias for DateTime)—while the corresponding parameter of String.Format is expecting an Object, the compiler also wraps the interpolation expression with a Convert node to Object.

Extension Method Calls These are actually calls to module-level methods (static methods in C#), and they're rendered as such in expression trees. Module-level and Shared methods don't have an instance, so the corresponding MethodCallExpression's Object property will return Nothing (or null). If the MethodCallExpression represents an instance method call, the Object property will not be Nothing.

The way extension methods are represented in expression trees highlights an important difference between expression trees and Roslyn compiler syntax trees, which preserve the syntax as is; expression trees are focused less on the precise syntax and more on the underlying operations. The Roslyn syntax tree for an extension method call would look the same as a standard instance method call, not a Shared or static method call.

Conversions When an expression's type doesn't match the expected type, and there's an implicit conversion from the expression's type, the compiler will wrap the inner expression in a Convert node to the expected type. The Visual Basic compiler will do the same when generating expression trees with an expression whose type implements or inherits from the expected type. For example, in **Figure 4**, the Count extension method expects an IEnumerable(Of Char), but the actual type of the expression is String.

Constructing Expression Tree Objects II: Using the Factory Methods

You can also construct expression trees using the Shared (static in C#) factory methods at System.Linq.Expressions.Expression. For example, to construct the expression tree objects for i.ToString, where i is an Integer in Visual Basic (or an int in C#), use code like the following:

```
' Imports System.Linq.Expressions.Expression

Dim prm As ParameterExpression = Parameter(GetType(Integer),
"i")
Dim expr As Expression = [Call](
  prm,
  GetType(Integer).GetMethods("ToString", {})
)
```

Meta Node Types

Most node types represent code operations. However, there are three "meta" node types—node types that provide information about the tree, but do not map directly to code.

ExpressionType.Quote This type of node always wraps a LambdaExpression, and specifies that the LambdaExpression defines a new expression tree, not a delegate. For example, the tree generated from the following Visual Basic code:

```
Dim expr As Expression(Of Func(Of Func(Of Boolean))) = Function()
Function() True
```

or the following C# code:

```
Expression<Func<Func<bool>>> expr = () => () => true;
```

represents a delegate that produces another delegate. If you want to represent a delegate that produces another expression tree, you have to wrap the inner one in a Quote node. The compiler does this automatically with the following Visual Basic code:

```
Dim expr As Expression(Of Func(Of Expression(Of Func(Of Boolean)))) =
  Function() Function() True
```

Or the following C# code:

```
Expression<Func<Expression<Func<bool>>>> expr = () => () => true;
```

ExpressionType.DebugInfo This node emits debug information, so when you debug compiled expressions, the IL at a particular point can be mapped to the right place in your source code.

ExpressionType.RuntimeVariablesExpression Consider the arguments object in ES3; it's available within a function without having been declared as an explicit variable. Python exposes a locals function, which returns a dictionary of variables defined in the local namespace. These "virtual" variables are described within an expression tree using the RuntimeVariablesExpression.

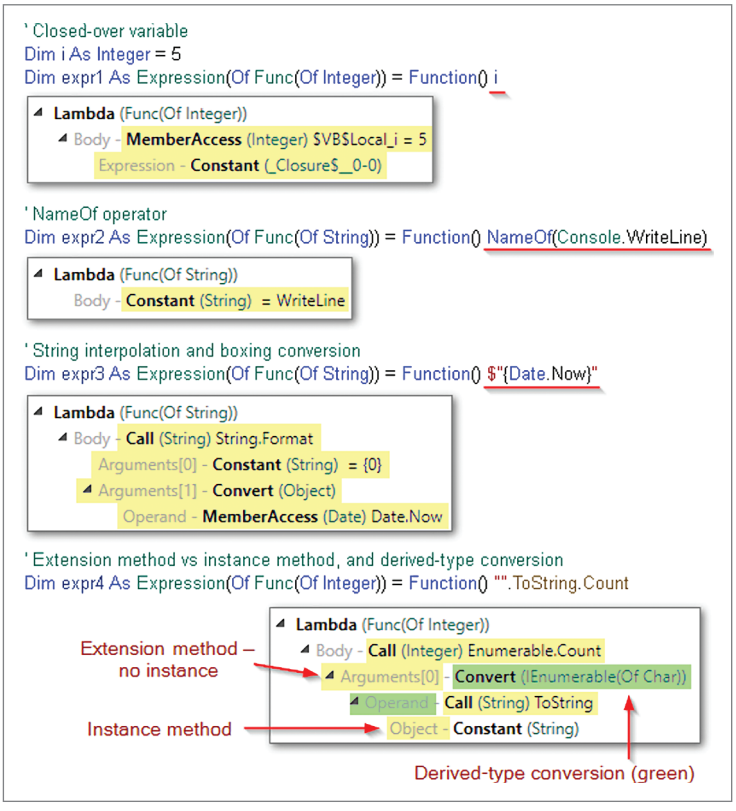


Figure 4 **Compiler-Generated Trees vs. Source Code**

In C#, the code should look like this:

```
// Using static System.Linq.Expressions.Expression

ParameterExpression prm = Parameter(typeof(int), "i");
Expression expr = Call(
  prm,
  typeof(int).GetMethods("ToString", new [] {})
);
```

While building expression trees in this manner usually requires a fair amount of reflection and multiple calls to the factory methods, you have greater flexibility to tailor the precise expression tree needed. With the compiler syntax, you would have to write out all possible variations of the expression tree that your program might ever require.

Closed-over Variables

When a lambda expression references a variable defined outside of it, we say the lambda expression "closes over" the variable. The compiler has to give this variable some special attention, because the lambda expression might be used after the value of the variable has been changed, and the lambda expression is expected to reference the new value. Or vice versa, the lambda expression might change the value, and that change should be visible outside of the lambda expression. For example, in the following C# code:

```
var i = 5;
Action lmbd = () => Console.WriteLine(i);
i = 6;
lmbd();
```

or in the following Visual Basic code:

```
Dim i = 5
Dim lmbd = Sub() Console.WriteLine(i)
i = 6
lmbd()
```

the expected output would be 6, not 5, because the lambda expression should use the value of i at the point when the lambda expression is invoked, after i has been set to 6.

The C# compiler accomplishes this by creating a hidden class with the needed variables as fields of the class, and the lambda expressions as methods on the class. The compiler then replaces all references to that variable with member access on the class instance. The class instance doesn't change, but the value of its fields can. The resulting IL looks something like the following in C#:

```
[CompilerGenerated]
private sealed class <>c__DisplayClass0_0 {
  public int i;
  internal void <Main>b__0() => Console.WriteLine(i);
}

var @object = new <>c__DisplayClass0_0();
@object.i = 5;
Action lmbd = @object.<Main>b__0;
@object.i = 6;
lmbd();
```

The Visual Basic compiler does something similar, with one difference: VBLocal is prepended to the property name, like so:

```
<CompilerGenerated> Friend NotInheritable Class _Closure$__0-0
  Public $VB$Local_i As Integer
  Sub _Lambda$__0()
    Console.WriteLine($VB$Local_i)
  End Sub
End Class

Dim targetObject = New _Closure$__0-0 targetObject With { .$VB$Local_i = 5 }
Dim lmbd = AddressOf targetObject. _Lambda$__0
targetObject.i = 6
lmbd()
```

In addition, the factory method API allows you to build some kinds of expressions that aren't currently supported in compiler-generated expressions. A few examples:

Statements such as a System.Void-returning Conditional-Expression, which corresponds to If..Then and If..Then..Else..End If (in C# known as if (...) { ...} else { ... }); or a TryCatchExpression representing a Try..Catch or a try { ... } catch (...) { ... } block.

Assignments Dim x As Integer: x = 17.

Blocks for grouping multiple statements together.

For example, consider the following Visual Basic code:

```
Dim msg As String = "Hello!"
If DateTime.Now.Hour > 18 Then msg = "Good night"
Console.WriteLine(msg)
```

or the following equivalent C# code:

```
string msg = "Hello";
if (DateTime.Now.Hour > 18) {
  msg = "Good night";
}
Console.WriteLine(msg);
```

You could construct a corresponding expression tree in Visual Basic or in C# using the factory methods, as shown in **Figure 5**.

Figure 5 Blocks, Assignments and Statements in Expression Trees

```
' Visual Basic
' Imports System.Linq.Expressions.Expression

Dim msg = Parameter(GetType(String), "msg")
Dim body = Block(
  Assign(msg, Constant("Hello")),
  IfThen(
    GreaterThan(
      MakeMemberAccess(
        MakeMemberAccess(
          Nothing,
          GetType(DateTime).GetMember("Now").Single
        ),
        GetType(DateTime).GetMember("Hour").Single
      ),
      Constant(18)
    ),
    Assign(msg, Constant("Good night"))
  ),
  [Call](
    GetType(Console).GetMethod("WriteLine", { GetType(string) }),
    msg
  )
)

// C#
// Using static System.Linq.Expressions.Expression

var msg = Parameter(typeof(string), "msg");
var expr = Lambda(
  Block(
    Assign(msg, Constant("Hello")),
    IfThen(
      GreaterThan(
        MakeMemberAccess(
          MakeMemberAccess(
            null,
            typeof(DateTime).GetMember("Now").Single()
          ),
          typeof(DateTime).GetMember("Hour").Single()
        ),
        Constant(18)
      ),
      Assign(msg, Constant("Good night"))
    ),
    Call(
      typeof(Console).GetMethod("WriteLine", new[] { typeof(string) }),
      msg
    )
  )
);
```

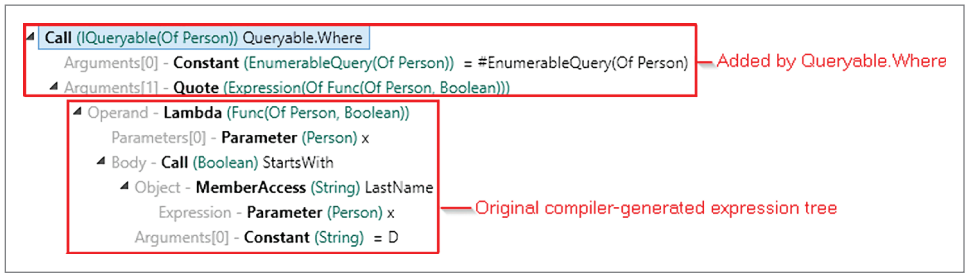


Figure 6 **Visualization of Final Expression Tree**

Using Expression Trees I: Mapping Code Constructs to External APIs

Expression trees were originally designed to enable mapping of Visual Basic or C# syntax into a different API. The classic use case is generating a SQL statement, like this:

```
SELECT * FROM Persons WHERE Persons.LastName LIKE N'D%'
```

This statement could be derived from code like the following snippet, which uses member access (. operator), method calls inside the expression, and the Queryable.Where method. Here's the code in Visual Basic:

```
Dim personSource As IQueryable(Of Person) = ...
Dim qry = personSource.Where(Function(x) x.LastName.StartsWith("D"))
```

and here's the code in C#:

```
IQueryable<Person> personSource = ...
var qry = personSource.Where(x => x.LastName.StartsWith("D");
```

How does this work? There are two overloads that could be used with the lambda expression—Enumerable.Where and Queryable.Where. However, overload resolution prefers the overload in which the lambda expression is an expression lambda—that is, Queryable.Where—over the overload that takes a delegate. The compiler then replaces the lambda syntax with calls to the appropriate factory methods.

At runtime, the Queryable.Where method wraps the passed-in expression tree with a Call node whose Method property references Queryable.Where itself, and which takes two parameters—personSource, and the expression tree from the lambda syntax (**Figure 6**). (The Quote node indicates that the inner expression tree is being passed to Queryable.Where as an expression tree and not as a delegate.)

A LINQ database provider (such as Entity Framework, LINQ2SQL or NHibernate) can take such an expression tree and map the different parts into the SQL statement at the beginning of this section. Here's how:

- ExpresssionType.Call to Queryable.Where is parsed as a SQL WHERE clause
- ExpressionType.MemberAccess of LastName on an instance of Person becomes reading the LastName field in the Persons table—Persons.LastName
- ExpressionType.Call to the StartsWith method with a Constant argument is translated into the SQL LIKE operator, against a pattern that matches the beginning of a constant string:

```
LIKE N'D%'
```

It's thus possible to control external APIs using code constructs and conventions, with all the benefits of the compiler—type safety and correct syntax on the various parts of the expression tree, and IDE autocompletion. Some other examples:

Creating Web Requests Using the Simple.OData.Client library (bit.ly/2YyDrsx), you can create OData requests by passing in expression trees to the various methods. The library will output the correct request (**Figure 7** shows the code for both Visual Basic and C#).

Reflection by Example Instead of using reflection to get hold of a MethodInfo, you can write a method call within an expression, and pass that expression to a function that extracts the specific overload used in the call. Here's the reflection code first in Visual Basic:

```
Dim writeLine as MethodInfo = GetType(Console).GetMethod(
  "WriteLine", { GetType(String) })
```

And the same code in C#:

```
MethodInfo writeLine = typeof(Console).GetMethod(
  "WriteLine", new [] { typeof(string) });
```

Now here's how this function and its usage could look in Visual Basic:

```
Function GetMethod(expr As Expression(Of Action)) As MethodInfo
  Return CType(expr.Body, MethodCallExpression).Method
End Function

Dim mi As MethodInfo = GetMethod(Sub() Console.WriteLine(""))
```

and here's how it could look in C#:

```
public static MethodInfo GetMethod(Expression<Action> expr) =>
  (expr.Body as MethodCallExpression).Method;

MethodInfo mi = GetMethod(() => Console.WriteLine(""));
```

This approach also simplifies getting at extension methods and constructing closed generic methods, as shown here in Visual Basic:

```
Dim wherePerson As MethodInfo = GetMethod(Sub() CType(Nothing, IQueryable(Of
  Person)).Where(Function(x) True))
```

It also provides a compile-time guarantee that the method and overload exist, as shown here in C#:

```
// Won't compile, because GetMethod expects Expression<Action>, not
Expression<Func<..>>
MethodInfo getMethod = GetMethod(() => GetMethod(() => null));
```

Grid Column Configuration If you have some kind of grid UI, and you want to allow defining the columns declaratively, you could

Figure 7 **Requests Using the Simple.OData.Client Library and Expression Trees**

```
' Visual Basic
Dim client = New ODataClient("https://services.odata.org/v4/
TripPinServiceRW/")
Dim people = Await client.For(Of People)
  .Filter(Function(x) x.Trips.Any(Function(y) y.Budget > 3000))
  .Top(2)
  .Select(Function(x) New With { x.FirstName, x.LastName})
  .FindEntriesAsync

// C#
var client = new ODataClient("https://services.odata.org/v4/
TripPinServiceRW/");
var people = await client.For<People>()
  .Filter(x => x.Trips.Any(y => y.Budget > 3000))
  .Top(2)
  .Select(x => new {x.FirstName, x.LastName})
  .FindEntriesAsync();

Outgoing request:
> https://services.odata.org/v4/TripPinServiceRW/People?$top=2 &amp;
  $select=FirstName, LastName &amp; $filter=Trips/any(d:d/Budget gt 3000)
```

have a method that builds the columns based on subexpressions in an array literal (using Visual Basic):

```
grid.SetColumns(Function(x As Person) {x.LastName, x.FirstName, x.DateOfBirth})
```

Or from the subexpressions in an anonymous type (in C#):

```
grid.SetColumns((Person x) => new {x.LastName, x.FirstName, DOB = x.DateOfBirth});
```

Using Expression Trees II: Compiling Invocable Code at Runtime

The second major use case for expression trees is to generate executable code at runtime. Remember the body variable from before? You can wrap it in a LambdaExpression, compile it into a delegate, and invoke the delegate, all at runtime. The code would look like this in Visual Basic:

```
Dim lambdaExpression = Lambda(Of Action)(body)
Dim compiled = lambdaExpression.Compile
compiled.Invoke
' prints either "Hello" or "Good night"
```

and like this in C#:

```
var lambdaExpression = Lambda<Action>(body);
var compiled = lambdaExpression.Compile();
compiled.Invoke();
// Prints either "Hello" or "Good night"
```

Compiling an expression tree into executable code is very useful for implementing other languages on top of the CLR, as it's much easier to work with expression trees over direct IL manipulation. But if you're programming in C# or Visual Basic, and you know the program's logic at compile time, why not embed that logic in your existing method or assembly at compile time, instead of compiling at runtime?

However, runtime compilation really shines when you don't know the best path or the right algorithm at design time. Using expression trees and runtime compilation, it's relatively easy to iteratively rewrite and refine your program's logic at runtime in response to actual conditions or data from the field.

Self-Rewriting Code: Call-Site Caching in Dynamically Typed Languages As an example, consider call-site caching in the Dynamic Language Runtime (DLR), which enables dynamic typing in CLR-targeting language implementations. It uses expression trees to provide a powerful optimization, by iteratively rewriting the delegate assigned to a specific call site when needed.

Both C# and Visual Basic are (for the most part) statically typed languages—for every expression in the language we can resolve a fixed type that doesn't change over the lifetime of the program. In other words, if the variables x and y have been declared as an Integer (or an int in C#) and the program contains a line of code x + y, the resolution of the value of that expression will always use the "add" instruction for two Integers.

However, dynamically typed languages have no such guarantee. Generally, x and y have no inherent type, so the evaluation of x + y must take into account that x and y could be of any type, say String, and resolving x + y in that case would mean using String.Concat. On the other hand, if the first time the program hits the expression x and y are Integers, it's highly likely that successive hits will also have the same types for x and y. The DLR takes advantage of this fact with *call-site caching*, which uses expression trees to rewrite the delegate of the call site each time a new type is encountered.

Each call site gets assigned a CallSite(Of T) (or in C# a CallSite<T>) instance with a Target property that points to a compiled delegate. Each site also gets a set of tests, along with the actions that should be taken when each test succeeds. Initially, the Target delegate only has code to update itself, like so:

```
' Visual Basic code representation of Target delegate
Return site.Update(site, x, y)
```

At the first iteration, the Update method will retrieve an applicable test and action from the language implementation (for example, "if both arguments are Integers, use the 'add' instruction"). It will then generate an expression tree that performs the action only if the test succeeds. A code-equivalent of the resulting expression tree might look something like this:

```
' Visual Basic code representation of expression tree
If TypeOf x Is Integer AndAlso TypeOf y Is Integer Then Return CInt(x) + CInt(y)
Return site.Update(site, x, y)
```

The expression tree will then be compiled into a new delegate, and stored at the Target property, while the test and action will be stored in the call site object.

In later iterations, the call site will use the new delegate to resolve x + y. Within the new delegate, if the test passes, the resolved CLR operation will be used. Only if the tests fail (in this case, if either x or y isn't an Integer) will the Update method have to turn again to the language implementation. But when the Update method is called it will add the new test and action, and recompile the Target delegate to account for them. At that point, the Target delegate will contain tests for all the previously encountered type pairs, and the value resolution strategy for each type, as shown in the following code:

```
If TypeOf x Is Integer AndAlso TypeOf y Is Integer Then Return CInt(x) + CInt(y)
If TypeOf x Is String Then Return String.Concat(x, y)
Return site.Update(site, x, y)
```

This runtime code rewriting in response to facts on the ground would be very difficult—if not impossible—without the ability to compile code from an expression tree at runtime.

Dynamic Compilation with Expression Trees vs. with Roslyn You can also dynamically compile code from simple strings rather than from an expression tree with relative ease using Roslyn. In fact, this approach is actually preferred if you're starting off with Visual Basic or C# syntax, or if it's important to preserve the Visual Basic or C# syntax you've generated. As noted earlier, Roslyn syntax trees model syntax, whereas expression trees only represent code operations without regard for syntax.

Also, if you try to construct an invalid expression tree, you'll get back only a single exception. When parsing and compiling strings to runnable code using Roslyn, you can get multiple pieces of diagnostic information on different parts of the compilation, just as you would when writing C# or Visual Basic in Visual Studio.

On the other hand, Roslyn is a large and complicated dependency to add to your project. You may already have a set of code operations that came from a source other than Visual Basic or C# source code; rewriting into the Roslyn semantic model may be unnecessary. Also, keep in mind that Roslyn requires multithreading, and cannot be used if new threads aren't allowed (such as within a Visual Studio debugging visualizer).

Rewriting Expression Trees: Implementing Visual Basic's Like Operator

I mentioned that expression trees are immutable; but you can create a new expression tree that reuses parts of the original. Let's imagine

Figure 8 **ExpressionTreeVisitor Replacing Visual Basic Like with DbFunctions.Like**

```
Class LikeVisitor
  Inherits ExpressionVisitor

  Shared LikeString As MethodInfo =
    GetType(CompilerServices.LikeOperator).GetMethod("LikeString")
  Shared DbFunctionsLike As MethodInfo = GetType(DbFunctions).GetMethod(
    "Like", {GetType(String), GetType(String)})

  Protected Overrides Function VisitMethodCall(
    node As MethodCallExpression) As Expression
    ' Is this node using the LikeString method? If not, leave it alone.
    If node.Method <> LikeString Then Return MyBase.VisitMethodCall(node)

    Dim patternExpression = node.Arguments(1)
    If patternExpression.NodeType = ExpressionType.Constant Then
      Dim oldPattern =
        CType(CType(patternExpression, ConstantExpression).Value, String)

      ' partial mapping of Visual Basic's Like syntax to SQL LIKE syntax
      Dim newPattern = oldPattern.Replace("*", "%")

      patternExpression = Constant(newPattern)
    End If

    Return [Call](DbFunctionsLike,
      node.Arguments(0),
      patternExpression
    )
  End Function
End Class
```

you want to query a database for people whose first name contains an "e" and a subsequent "i." Visual Basic has a Like operator, which returns True if a string matches against a pattern, as shown here:

```
Dim personSource As IQueryable(Of Person) = ...
Dim qry = personSource.Where(Function(x) x.FirstName Like "*e*i*")
For Each person In qry
  Console.WriteLine($"LastName: {person.LastName}, FirstName: {person.FirstName}")
Next
```

But if you try this on an Entity Framework 6 DbContext, you'll get an exception with the following message:

```
'LINQ to Entities does not recognize the method 'Boolean
LikeString(System.String, System.String, Microsoft.VisualBasic.
CompareMethod)' method, and this method cannot be translated into a store
expression.'
```

The Visual Basic Like operator resolves to the LikeOperator.LikeString method (in the Microsoft.VisualBasic.CompilerServices namespace), which EF6 can't translate into a SQL LIKE expression. Thus, the error.

Now EF6 does support similar functionality via the DbFunctions.Like method, which EF6 can map to the corresponding LIKE.

More Information

- Expression Trees in the programming guides for Visual Basic (bit.ly/2Msocef) and C# (bit.ly/2Y9q5nj)
- Lambda expressions in the programming guides for Visual Basic (bit.ly/2YsZFs3) and C# (bit.ly/331ZWp5)
- Bart De Smet's Expression Tree Futures project (bit.ly/2OrUsRw)
- DLR project on GitHub (bit.ly/2yssz0x) has documents describing the design of expression trees in .NET
- Rendering expression trees as strings, and debugging visualizers for expression trees—bit.ly/2MsoXnB and bit.ly/2GAp5ha
- "What does Expression.Quote() do that Expression.Constant() can't already do?" on StackOverflow (bit.ly/30YT6Pi)

You need to replace the original expression tree, which uses the Visual Basic Like, with one that uses DbFunctions.Like, but without changing any other parts of the tree. The common idiom for doing this is to inherit from the .NET ExpressionVisitor class, and override the Visit* base methods of interest. In my case, because I want to replace a method call, I'll override VisitMethodCall, as shown in **Figure 8**.

The Like operator's pattern syntax is different from that of SQL LIKE, so I'll replace the special characters used in the Visual Basic Like with the corresponding ones used by SQL LIKE. (This mapping is incomplete—it doesn't map all the pattern syntax of the Visual Basic Like; and it doesn't escape SQL LIKE special characters, or unescape Visual Basic Like special characters. A full implementation can be found on GitHub at bit.ly/2yku7tx, together with the C# version.)

Note that I can only replace these characters if the pattern is part of the expression tree, and that the expression node is a Constant. If the pattern is another expression type—such as the result of a method call, or the result of a BinaryExpression that concatenates two other strings—then the value of the pattern doesn't exist until the expression has been evaluated.

I can now replace the expression with the rewritten one, and use the new expression in my query, like so:

```
Dim expr As Expression(Of Func(Of Person, Boolean)) =
  Function(x) x.FirstName Like "*e*i*"
Dim visitor As New LikeVisitor
expr = CType(visitor.Visit(expr), Expression(Of Func(Of Person, Boolean)))

Dim personSource As IQueryable(Of Person) = ...
Dim qry = personSource.Where(expr)
For Each person In qry
  Console.WriteLine($"LastName: {person.LastName}, FirstName: {person.FirstName}")
Next
```

Ideally, this sort of transformation would be done within the LINQ to Entities provider, where the entire expression tree—which might include other expression trees and Queryable method calls—could be rewritten in one go, instead of having to rewrite each expression before passing it into the Queryable methods. But the transformation at its core would be the same—some class or function that visits all the nodes and plugs in a replacement node where needed.

Wrapping Up

Expression trees model various code operations and can be used to expose APIs without requiring developers to learn a new language or vocabulary—the developer can leverage Visual Basic or C# to drive these APIs, while the compiler provides type-checking and syntax correctedness, and the IDE supplies Intellisense. A modified copy of an expression tree can be created, with added, removed, or replaced nodes. Expression trees can also be used for dynamically compiling code at runtime, and even for self-rewriting code; even as Roslyn is the preferred path for dynamic compilation.

Code samples for this article can be found at bit.ly/2yku7tx. ■

Zev Spitz *has written a library for rendering expression trees as strings in multiple formats—C#, Visual Basic and factory method calls; and a Visual Studio debugging visualizer for expression trees.*

Thanks to the following Microsoft technical expert for reviewing this article: *Kathleen Dollard*

Manipulating Documents?

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

Try GroupDocs APIs for FREE

Download a Free Trial at

https://downloads.groupdocs.com

GROUPDOCS

GroupDocs.Viewer

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

GroupDocs.Annotation

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

GroupDocs.Conversion

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

GroupDocs.Comparison

Compare two documents and get a difference summary report.

GroupDocs.Signature

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

GroupDocs.Assembly

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

GroupDocs.Metadata

Organize documents with metadata within any cross platform application.

GroupDocs.Search

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

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

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

DINO ESPOSITO

Streaming Methods in ASP.NET Core gRPC Services

In the previous installment of Cutting Edge, I walked through building a new type of service based on the gRPC framework that (although available to C# developers for a while) in ASP.NET Core 3.0 debuts as a native service hosted directly by Kestrel. The gRPC framework is suited for peer-to-peer binary communication between connected endpoints—mostly, but not necessarily, microservices. It also supports up-to-date technical solutions such as Google Protobuf for serialization of content and HTTP/2 for transportation.

Visual Studio 2019 comes with an ASP.NET Core 3.0 project template for creating the skeleton of a gRPC service in just a few clicks. For a primer both about gRPC and the starter kit generated by Visual Studio, you can check out my July column at msdn.com/magazine/mt833481. This month I take my exploration of gRPC one step further. First, I'll cover the underlying tooling in a bit more detail. In fact, some tooling is necessary to parse the content of the .proto file to C# classes to be used as the foundation of both client and service implementations. In addition, I'll touch on streaming methods and complex message classes. Finally, I'll focus on how to integrate streamed gRPC methods in the UI of a Web client application.

Building the gRPC Service

The built-in Visual Studio project template places the service interface definition file (the .proto file) in a subfolder named protos located in the same service project. In this article, though, I'll take a different approach and start by adding a brand new .NET Standard 2.0 class library to the initially empty solution.

The proto class library doesn't explicitly contain any C# class. All it contains is one or more .proto files. You can organize .proto files in folders and subfolders at your leisure. In the sample application, I have a single .proto file for a sample service located under the protos project folder. Here's an excerpt from the service block of the sample .proto file:

```
service H2H {
  rpc Details (H2HRequest) returns (H2HReply) {}
}
```

The H2H sample service is expected to retrieve some sport-related information from some remote location. The Details method passes a head-to-head request and receives the score of the past matches between specified players or teams. Here's what the H2HRequest and H2HReply messages may look like:

Code download available at msdn.com/magazine/0919magcode.

```
message H2HRequest {
  string Team1 = 1;
  string Team2 = 2;
}
message H2HReply {
  uint32 Won1 = 1;
  uint32 Won2 = 2;
  bool Success = 3;
}
```

The first message type passes information about the teams to process, whereas the latter receives the history of past matches and a Boolean flag that denotes success or failure of the operation. So far so good. Everything in the messages is defined as we've seen in the previous article. Using the gRPC jargon, the Details method is a unary method, meaning that every request receives one (and only one) response. This is the most common way of coding a gRPC service, however. Let's add streaming capabilities, like so:

```
rpc MultiDetails (H2HMultiRequest) returns (stream H2HMultiReply) {}
```

> You can organize .proto files in folders and subfolders at your leisure.

The new MultiDetails method is a server-side streaming method, meaning that for each request it gets from some gRPC client it may return multiple responses. In this example, the client might send an array of head-to-head requests and receive individual head-to-head responses in an asynchronous way as they're elaborated on the service end. For this to happen, the gRPC service method must be labeled with the stream keyword in the returns section. A stream method may require ad hoc message types, too, as shown here:

```
message H2HMultiRequest {
  string Team = 1;
  repeated string OpponentTeam = 2;
}
```

As mentioned, the client may ask for a head-to-head record between one particular team and an array of other teams. The repeated keyword in the message type just denotes that the OpponentTeam member may appear more than once. In pure C# terms, the H2HMultiRequest message type is conceptually equivalent to the following pseudocode:

```
class H2HMultiRequest
{
  string Team {get; set;}
  IEnumerable<string> OpponentTeam {get; set;}
}
```

However, note that the code generated by the gRPC tooling is slightly different, as shown here:

```
public RepeatedField<string> OpponentTeam {get; set;}
```

Note, in fact, that any class generated from a gRPC message type T implements the members of the Google.ProtoBuf.IMessage<T> interface. The response message type should be designed to describe the actual data returned at each step of the streaming phase. So each reply must refer to an individual head-to-head response, between the primary team and one of the opponent teams specified in the array, like so:

```
message H2HMultiReply {
  H2HItem Team1 = 1;
  H2HItem Team2 = 2;
}

message H2HItem {
  string Name = 1;
  uint32 Won = 2;
}
```

The H2HItem message type indicates how many matches the given team has won against the other team specified in the request.

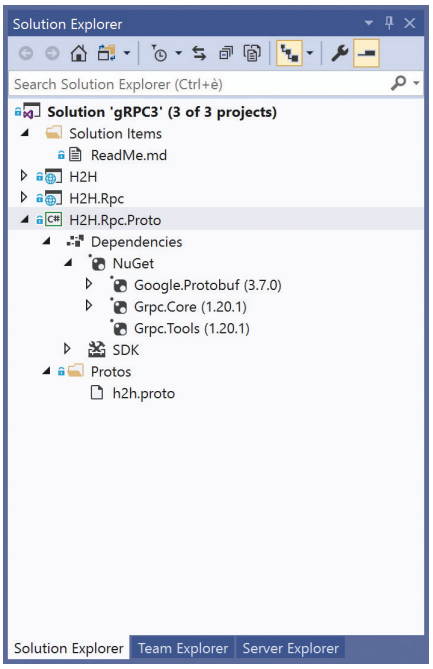


Figure 1 **The NuGet Dependencies of the Proto Shared Class Library**

Before I move forward to look into the implementation of a stream method, let's have a look at the dependencies required by the shared class library that embeds the proto definition. The Visual Studio project must reference the NuGet packages in **Figure 1**.

The project that includes the source .proto file must reference the Grpc.Tools package, as well as the Grpc.Net.Client (added in .NET Core 3.0 Preview 6) and Google.Protobuf packages required by any gRPC project (whether client, service or library). The tooling package is ultimately responsible for parsing the .proto file and generating any necessary C# classes at compile time. An item group block in the .csproj file instructs the tooling system on how to proceed. Here's the code:

```
<ItemGroup>
  <Protobuf Include="Protos\h2h.proto"
            GrpcServices="Server, Client"
            Generator="MSBuild:Compile" />
  <Content Include="@(Protobuf)" />
  <None Remove="@(Protobuf)" />
</ItemGroup>
```

The most relevant part of the ItemGroup block is the Protobuf node and in particular the GrpcServices attribute. The Server token in its assigned string value indicates that the tooling must generate the service class for the prototyped interface. The Client token indicates that it's also expected to create the base client class to invoke the service. With this done, the resulting DLL contains C# classes for the message types, base service class and client class. Both the service project and the client project (whether console, Web or desktop) only need to reference the prototype DLL in order to deal with the gRPC service.

Figure 2 **Configuring the gRPC Service**

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddGrpc();
}
public void Configure(IApplicationBuilder app)
{
  // Some other code here
  ...

  app.UseRouting();
  app.UseEndpoints(endpoints =>
  {
    endpoints.MapGrpcService<H2HService>();
  });
}
```

Implementing the Service

The gRPC service is an ASP.NET Core project with some special configuration done in the startup class. In addition to the ASP.NET Core server platform and the prototype assembly, it references also the ASP.NET Core gRPC framework and the Google.Protobuf package. The Startup class adds the gRPC runtime service in the Configure method and appends gRPC endpoints in the ConfigureServices method, as shown in **Figure 2**.

The service class inherits from the base service class the tooling created based on the content of the .proto file, like so:

```
public class H2HService : Sample.H2H.H2HBase
{
  // Unary method Details
  public override Task<H2HReply> Details(
              H2HRequest request, ServerCallContext context)
  {
    ...
  }

  ...
}
```

Unary methods like the Details method have a simpler signature than stream methods. They return a Task<TReply> object and accept a TRequest object plus an instance of ServerCallContext to access the nitty-gritty details of the incoming request. A server-side stream method has an additional response stream parameter used by the implementation code to stream packets back. **Figure 3** presents the implementation of the MultiRequest stream method.

As you can see, compared to classic unary methods, the stream method takes an additional parameter of type IServerStreamWriter<TReply>. This is the output stream the method will use to stream back results as they're ready. In the code in **Figure 3**, the method enters in a loop for each of the operations requested (in this case, an array of teams to get past matches). It then streams back results as the query against a local/remote database or Web service returns. Once complete, the method returns and the underlying runtime environment closes the stream.

Writing a Client for a Stream Method

In the sample code, the client application is a plain ASP.NET Core 3.0 application. It holds references to the Google.Protobuf package and the Grpc.Net.Client package, plus the shared prototype library. The UI presents a button with some JavaScript attached that posts

Figure 3 **A Server-Side Stream gRPC Service Method**

```
public override async Task MultiDetails(
      H2HMultiRequest request,
      IServerStreamWriter<H2HMultiReply> responseStream,
      ServerCallContext context)
{
  // Loops through the batch of operations embedded
  // in the current request
  foreach (var opponent in request.OpponentTeam)
  {
    // Grab H2H data to return
    var h2h = GetHeadToHead_Internal(request.Team, opponent);

    // Copy raw data into an official reply structure
    // Raw data is captured in some way: an external REST service
    // or some local/remote database
    var item1 = new H2HItem {
     Name = h2h.Id1, Won = (uint) h2h.Record1};
    var item2 = new H2HItem {
     Name = h2h.Id2, Won = (uint) h2h.Record2};

    var reply = new H2HMultiReply { Team1 = item1, Team2 = item2 };

    // Write back via the output response stream
    await responseStream.WriteAsync(reply);
  }

  return;
}
```

Figure 4 **Calling the gRPC Service**

```
[HttpPost]
public async Task<IActionResult> Multi()
{
  // Call the RPC service
  var serviceUrl = "http://localhost:50051";

    AppContext.SetSwitch(
               "System.Net.Http.SocketsHttpHandler.Http2UnencryptedSupport",
               true);
    var httpClient = new HttpClient() {BaseAddress = new Uri(serviceUrl) };
  var client = GrpcClient.Create<H2H.H2HClient>(httpClient);

  var request = new H2HMultiRequest() { Team = "AF-324" };
  request.OpponentTeam.AddRange(new[] { "AW-367", "AD-683", "AF-510" });
  var model = new H2HMultiViewModel();
  using (var response = client.MultiDetails(request))
  {
    while (await response.ResponseStream.MoveNext())
    {
      var reply = response.ResponseStream.Current;

      // Do something here ...
    }
  }
  return View(model);
}
```

to a controller method. (Note that nothing stops you from using a classic HTML Form except that using Ajax to post might make it easier to receive notification of replies and update the UI in a smoother way.) **Figure 4** presents the code.

packet from the gRPC service. What happens next depends on the client application. Overall, there are three distinct situations.

One is when the client application has its own UI, but can wait to collect the entire response before showing something fresh to the user. In this case, you load the data carried by the current reply object into the view model returned by the controller method. The second scenario is when there's no UI (such as if the client is a working microservice). In this case, the received data is processed as soon as it's available. Finally, in the third scenario the client application has its own responsive UI and is able to present

> It's worth recalling that the port of the gRPC service depends on the Visual Studio project, while the client caller class is defined in the prototype library.

It's worth recalling that the port of the gRPC service depends on the Visual Studio project, while the client caller class is defined in the prototype library. To prepare the request to a server-side streaming method, you don't have to do anything more than just populate the input message type. As mentioned, the OpponentTeam collection is an enumerable .NET type and can be populated with AddRange or repeated calls to Add. The actual type isn't one of the .NET Core collection types, but it's still a collection type despite being implemented in the Google Protobuf package.

As a server-side method streams packets back until the end of the stream, the actual call to the method returns a stream object. Next, the client code enumerates the packets waiting for the end of the response. Each iteration of the while loop in **Figure 4** captures a single reply

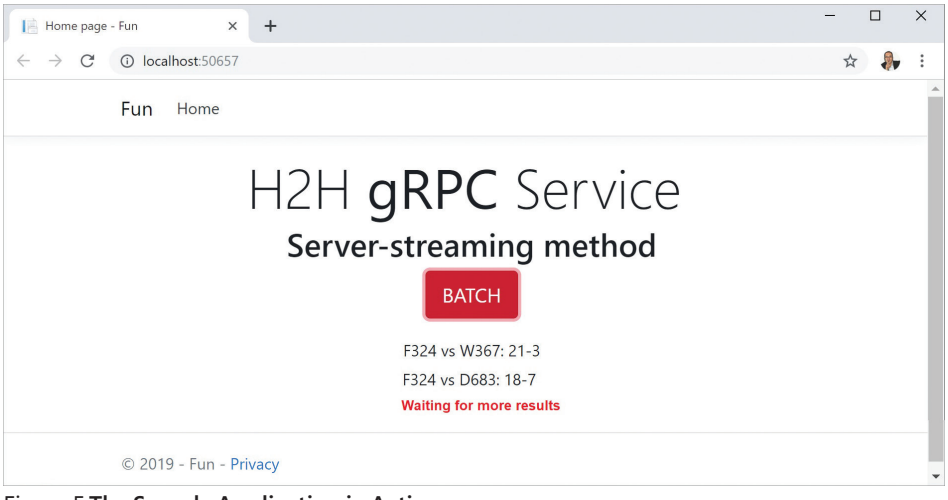


Figure 5 **The Sample Application in Action**

data to users as it comes from the server. In this case, you can attach a SignalR Core endpoint to the client application and notify the UI in real time (see **Figure 5**).

The following code snippet shows how the client code changes when a SignalR hub is used on top of the gRPC call:

```
var reply = response.ResponseStream.Current;
await _h2hHubContext.Clients
                    .Client(connId)
                    .SendAsync("responseReceived",
        reply.Player1.Name,
        reply.Player1.Won,
        reply.Player2.Name,
        reply.Player2.Won);
```

You can check the source code for full details of the solution. Speaking of SignalR, there are a couple of points worth exploring. First, SignalR code is used only by the client application that connects to the gRPC service. The hub is injected in the controller of the client application, not in the gRPC service. And second, as far as streaming is concerned, it's worth noting that SignalR Core also has its own streaming API.

Other Types of gRPC Streams

In this article I focused on server-side gRPC streaming methods, but that isn't the only option. The gRPC framework also supports client-side streaming methods (multiple requests/one response) and bidirectional streaming (multiple requests/multiple responses). For client-side streaming, the only difference is the use of a IAsyncStreamReader as the input stream in the service method, as shown in this code:

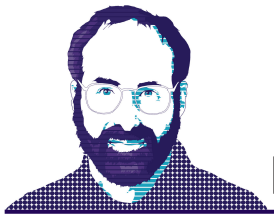
```
public override async Task<H2HReply> Multi(
        IAsyncStreamReader<H2HRequest> requestStream,
        ServerCallContext context)
{
  while (await requestStream.MoveNext())
  {
    var requestPacket = requestStream.Current;

      // Some other code here
      ...
  }
}
```

A bidirectional method will return void and take no parameters as it'll be reading and writing input and output data through input and output streams.

In summary, gRPC is a complete framework to connect two endpoints (client and server) over a binary, flexible and open source protocol. The support you get for gRPC from ASP.NET Core 3.0 is amazing and will improve over time, making now a great time to get started and experiment with gRPC, especially in microservice-to-microservice communication. ■

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

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



A C Change

Full fathom five thy father lies;
Of his bones are coral made;
Those are pearls that were his eyes;
Nothing of him that doth fade,
But doth suffer a sea-change
Into something rich and strange.
—Shakespeare, *The Tempest*, Act I, Scene ii

I've been around this industry a long time, and have seen many changes: some good, some bad, some undoing earlier ones. One of the biggest changes I've noticed from Microsoft over the years is the company's pivot toward open source, as many other software companies are also doing.

I remember when source code was the most closely guarded secret at Microsoft. I was allowed to see some back in 2004 when I did some work for Microsoft, related to the insurance industry. After pledging my firstborn as security against disclosing it, they showed me a file, so I could see where my code would hook into it. "OMG, the sacred Microsoft source. I'll never vacuum this laptop keyboard again!" (I think it still has some of Simba's fur in it.) I looked over my shoulder as I left the building, remembering the customary admonishment, "We could tell you, but then we'd have to kill you." So far, they haven't. Yet.

It's different now. At the Microsoft Build 2019 conference this spring, I caught up with my good friend Richard Turner, now the product manager of Microsoft Terminal. He told me how the new implementation was completely open source (repository at github.com/microsoft/terminal, and you can see his Build presentation at youtu.be/KMudkRcwjCw). I took a good look through the code. And you know what? The heavens didn't tremble. It was regular C++ code, similar to that produced by good programmers at other companies. I was especially struck by the comments, such as "Load Bearing Code—do not touch" or "We need to come back and fix this someday." Microsoft programmers encounter the same time and logical constraints as the rest of us, and they react in similar ways. Who'd-a thunk it?

Customers are demanding open source software, so vendors are supplying it, as classical economic theory would dictate. But this shift has ramifications that the purely rational model doesn't cover. Behavioral economist Dan Ariely discusses this in his excellent book, "Predictably Irrational: The Hidden Forces That Shape Our Decisions" (Harper Collins, 2008). In Chapter 4, "The Cost of Social Norms," he describes the predictable differences in human reactions to business transactions versus social transactions, even at the same price point. Offer your in-laws $50 for serving you Thanksgiving dinner, and your celebration next year will probably be a frozen dinner alone in front of the TV set. Spend that same $20 on flowers and a decent bottle of wine, and now you're in a completely different interaction model; one that works on social norms, to which classical theories do not apply. Ariely writes specifically about open source software:

"… you can post a problem about a bug on one of the bulletin boards, and see how fast someone, or often many people, will react to your request and fix the software—using their own leisure time. Could you pay for this level of service? Most likely. But if you had to hire people of the same caliber, they would cost you an arm and a leg. Rather, people in these communities are happy to give their time to society at large (for which they get the same social benefits we all get from helping a friend paint a room)."

I took a good look through the code. And you know what? The heavens didn't tremble. It was regular C++ code, similar to that produced by good programmers at other companies.

My friends in the Visual Basic 6 community have been clamoring over many years for exactly this. (See my previous columns at msdn.com/magazine/mt846730, msdn.com/magazine/dn745870 and msdn.com/magazine/jj133828.) I don't know whether this will happen. Even though it would be open source, such a project would still require a significant effort from Microsoft: for coordination, for extensive testing, for ensuring compatibility. Much as I personally wish the company would do it, I can see where Microsoft might consider it not worth the trouble. Still, the prevailing winds are finally blowing in this direction, so hope springs eternal.

The open source sea change will alter the future direction of this industry in ways that are hard to predict. The model isn't new, but its widespread prevalence in the commercial sector is. I'll be curious to see how it evolves over time. ■

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

Address the ELEPHANT IN THE ROOM

Bad address data costs you money, customers and insight.

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

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

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

See the Elephant in Your Business -

Name it and Tame it!

melissa

www.Melissa.com | 1-800-MELISSA

Free API Trials, Data Quality Audit & Professional Services.

LEADTOOLS®

MEDICAL WEB VIEWER

Now with 3D

The LEADTOOLS Medical Web Viewer is an OEM-ready web application that provides a platform-independent solution to display DICOM studies for all medical disciplines and modalities. The fully customizable application is lightweight, yet includes powerful features including 3D reconstruction, hanging protocols, touch-enabled window-leveling, annotations and more, all in a web-based zero-footprint solution.

Microsoft® .NET · macOS · iOS

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM